Australian Government
Department of Defence
Defence Science and
Technology Organisation

Modelling of Medium Access Control (MAC) Protocols for Mobile Ad-hoc Networks

Raymee Chau

Information Networks Division
Information Sciences Laboratory

DSTO-TN-0637

ABSTRACT

This technical note explains the design and modelling of two MAC protocols, Adaptive Generalized Transmission (AGENT) protocol and Collision-Avoidance Time Allocation (CATA) protocol, for mobile ad-hoc networks. These MAC models will be used to assist the analysis of performance of MAC protocols in the future Tactical Data Distribution Sub-system (TDDS), in support of Joint Project 2072 (Battlespace Communications System Land).

Published by

DSTO Information Sciences Laboratory
PO Box 1500
Edinburgh South Australia 5111 Australia

Telephone: (08) 8259 5555
Fax: (08) 8259 6567

AR-013-420
June 2005

APPROVED FOR PUBLIC RELEASE

Modelling of Medium Access Control (MAC) Protocols for Mobile Ad-hoc Networks

Executive Summary

In the next decade, the Tactical Data Distribution Sub-system (TDDS) is planned to be employed in the Battlespace Communication System for the Land forces. The TDDS is proposed to be an ad-hoc network, as it requires high mobility, capacity and reliability. Due to these demands, it is important to determine suitable Medium Access Control (MAC) protocols. This can be determined through the modelling, simulation and analysis of candidate MAC protocols.

The MAC protocols that have been modelled to this point are the Adaptive Generalized Transmission (AGENT) protocol and the Collision-Avoidance Time Allocation (CATA) protocol. This report addresses the design and implementation of these two MAC protocol models in OPNET. In addition, a stuffing algorithm is implemented in the MAC models to enable the assembly of a number of small packets into one larger packet to enhance the performance of the protocols. The report also explains the verification of the aforementioned MAC models.

The results obtained from this work provide an indication of how well each protocol performs in a structured ad-hoc network. The models of these MAC protocols will be applied to Headline 2000 and other data to investigate their performance in a TDDS in the near future.

Authors

Raymee Chau
Information Networks Division

Raymee Chau is a Research Engineer in Mobile Networks Group of the Defence Science and Technology Organisation's (DSTO) Information Networks Division. She joined DSTO after completing a Computer Science and Electrical Engineering Degree at the University of Melbourne. She is involved in research into tactical communication systems.

Contents

Abbreviations

AGENT	Adaptive Generalized Transmission
ARP	Address Resolution Protocol
CATA	Collision Avoidance Time Allocation
CMS	Control Mini-Slot
CSMA	Carrier Sense Multiple Access
CTS	Clear To Send
DMS	Data Mini-Slot
IP	Internet Protocol
LAN	Local Area Network
MAC	Medium Access Control
MACAW	Medium Access Protocol for Wireless LANs
MANET	Mobile Ad-hoc Network
NCTS	Not Clear To Send
NTS	Not To Send
RTS	Request To Send
SR	Slot Reservation
TDDS	Tactical Data Distribution Sub-system
TDMA	Time Division Multiple Access

1. Introduction

The work described in this report aims to assist the development of the Tactical Data Distribution Sub-system (TDDS) of the Battlespace Communication System (Land). The TDDS is designed to support the delivery of data (including situational awareness) for the command and control of combat troops in future land warfare, which requires high capacity, mobility and reliability. In order to meet these requirements, a mobile ad-hoc network (MANET) has been proposed as a potential technical solution.

In this report, we focus on the design and the implementation of the medium access control (MAC) protocols for ad-hoc networks. Their performance is analysed using OPNET[1], a network modelling and simulation software package. The MAC protocols that we implemented were:

- Adaptive Generalized Transmission (AGENT) protocol, developed by A.D. Myers, G.V. Zaruba and V.R. Syrotiuk [1]; and
- Collision-Avoidance Time Allocation (CATA) protocol, developed by Z. Tang and J.J. Garcia-Luna-Aceves [2].

Since these MAC protocols are designed for ad-hoc networks, they have the following properties:

- Collision avoidance;
- Collision avoidance to hidden terminals;
- Support for both point-to-point (unicast) and point-to-multipoint (broadcast) modes; and
- Fair allocation of capacity.

The purpose of modelling the MAC protocols is to allow us to compare the performance of the different MAC protocols in a TDDS. In order to achieve this, the protocols will be compared using the same network model and the same set of data obtained from the wargame Headline 2000. The details of the design and modelling of an ad-hoc network utilising the Headline 2000 data are given in [3].

The implementation of the MAC protocols described in this report uses the analysis of W.D. Blair's paper [4]. The current report details the design and integration of the network model, the node model and the MAC protocol models in addition to providing simulation results to verify the model implementations. In Section 2, the modelling and simulation of an ad-hoc wireless network model that is used to validate the MAC models is illustrated. The structure of an ad-hoc wireless node that is used in the network model is explained in Section 3. Section 4 details the design of the process models that is implemented, and Section 5 discusses the outcome observed from the simulations.

[1] OPNET is a registered trademark of OPNET Technologies, Inc.

2. Network Model

Network modelling in OPNET is achieved by inserting nodes (such as mobile stations, base stations, routers and servers) and links into the Network Editor to create a network model. Each node consists of processors, queues, transmitters and receivers can be created and modified using the Node Editor. A Process Editor can be used to build the process models based on state diagrams.

Network modelling and simulation is accomplished in the Network Editor. Generally, a network is modelled in the network model using a series of node models and link models. As wireless ad-hoc networking is the focus of this work, a network with mobile nodes and radio links is modelled. This is shown in Figure 1. The communication between nodes is made via the radio channels. This network model is used to simulate and validate the MAC protocols, AGENT and CATA, which will be described in Section 4.

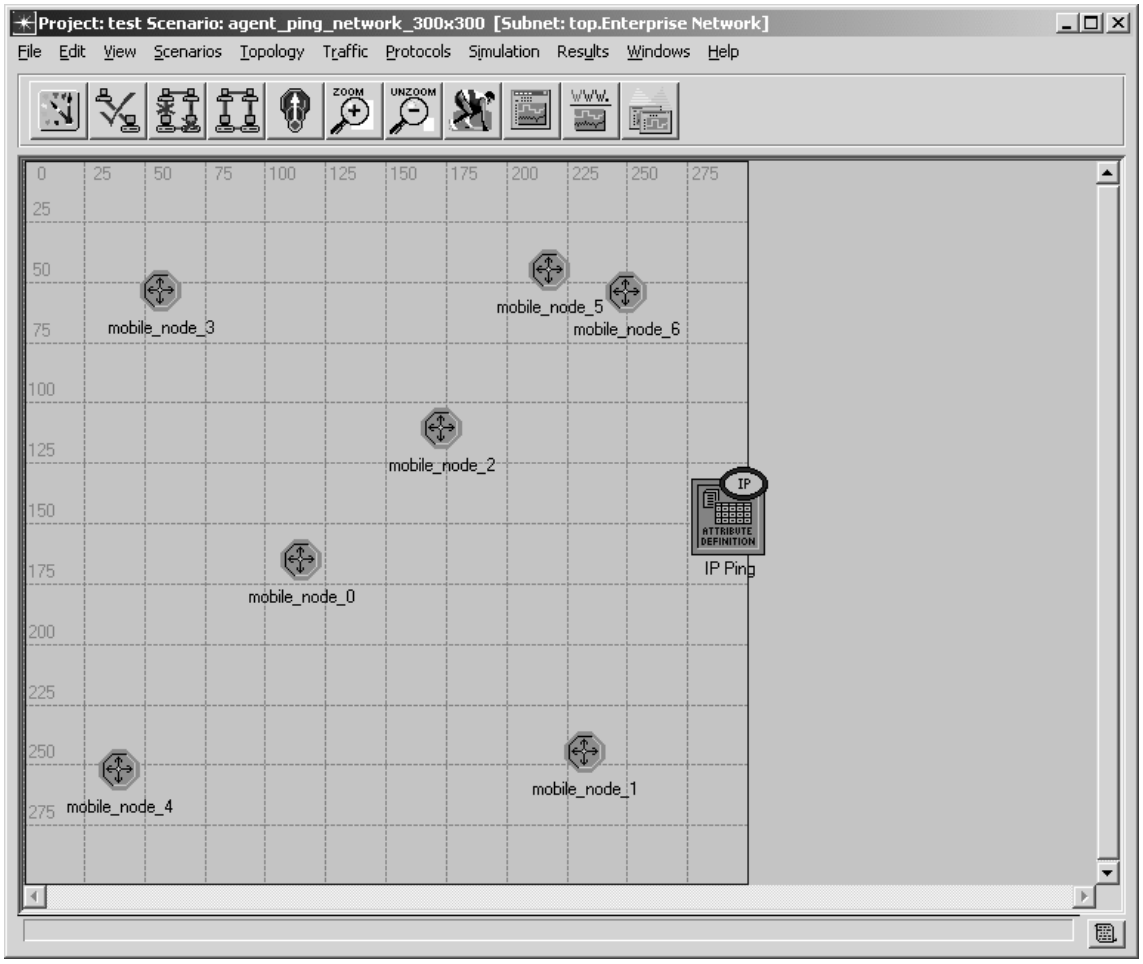

Figure 1: A network model

This network model is made up of a number of MANET nodes, and is designed to examine the functions of the MAC layer process models by transmitting packets between the nodes. To check that the MAC layer process models are adequate for representation of typical deployment, we set the size of the network model to 300x300km and the transmission rate of a node to 512 kbps[2]. The size of this network model is chosen so that the node at one corner is hidden from the node at its opposite corner (i.e. mobile_node_1 cannot hear mobile_node_3 and vice versa, because they are out of range). This can examine the hidden node collision avoidance ability of the MAC protocols. However, the ping traffic of the nodes must be configured so that no multi-hopping is required to transmit the packets from one node to any other node in the network, as a routing mechanism is not implemented in the node model.

In the network above, there is another node named *IP ping*. It is an Internet Protocol (IP) attribute configuration model. This model is used for setting the ping parameters to create the packets in the network layer. *IP ping* is applied because this network is designed to verify the functions of the MAC layer process models, the time required to simulate the network can be minimised by generating the packets in the IP layer rather than from the application layer.

The functions of each node in the network are implemented in the node model. The details about the functions of the nodes will be explained in Section 3.

3. Node Design

The nodes in the network model are modelled by using the Node Editor, a node modelling tool, in OPNET. They are constructed by connecting the processor, transmitter and receiver modules via the *packet stream links* or the *statistic wires*. The *packet stream links* are used to transfer the packets between the processors, and the *statistic wires* are used to transmit individual statistic values for interrupts.

A MANET node is modelled by linking specific modules via the *packet streams*. The modules that are used in this node model are shown in Figure 2. The major components of this node model includes:

- The IP and ARP models in the network layer;
- The MAC model in the link layer;
- A radio transmitter; and
- A radio receiver.

Although the node model has an application layer and a transport layer, the functions of these layers are not used in the simulations of this work. This is because IP packets will be

[2] These dimensions were found to be suitable. If the network model is too large, it's more complex to test, and if the network model is too small, we would be unable to test the cases where a node cannot be reached.

generated in the simulation rather than the packets from the application layer. However, these two layers must be included for the simulation to run.

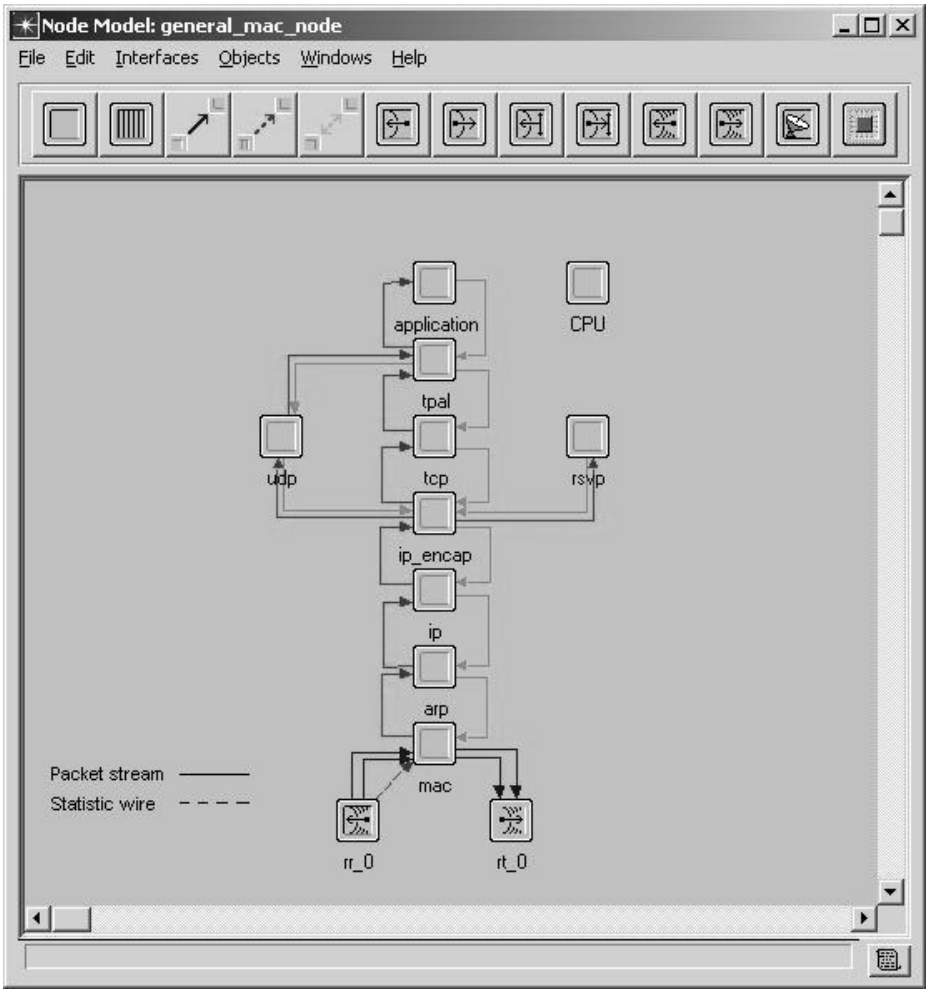

Figure 2: The node model of an ad-hoc node

This node model is created by modifying the Wireless LAN node model, *wlan_wkstn_adv*, provided in the OPNET package. The modifications of this node model include:

- Changing the process model of the *mac* process from *wireless_lan_mac* to *general_mac*. The process model, *general_mac*, is a newly implemented MAC protocol, which will be explained in Section 4.
- Modify the data rate and frequency of the radio transmitter and receiver.

The reason for reusing the existing Wireless LAN node model for this Ad-Hoc Network is because for this work, we are only required to test the MAC protocols, not the routing capability, and a MANET node without routing is like a Wireless LAN node. Thus, it is not necessary to implement a real MANET node with routing ability.

Below the MAC module, there is a radio transmitter and a radio receiver. In order to guarantee a valid connection between the nodes, it is important to ensure:

- A data rate of 512kbps in the radio transmitter and the radio receiver; and
- The bandwidths and the frequencies, which are referred to as "minimum frequency" in the transmitter and receiver attributes, are the same for both radio transmitter and radio receiver.

4. MAC Module

The MAC module consists of a generic MAC process model and one of the MAC protocol process models (i.e. AGENT or CATA). The structure of the MAC module is illustrated in Figure 3. It is designed to interact with the IP/ARP module (as shown in Figure 2). However, it is also possible for it to be integrated into a self-implemented higher layer module. This is explained in [3].

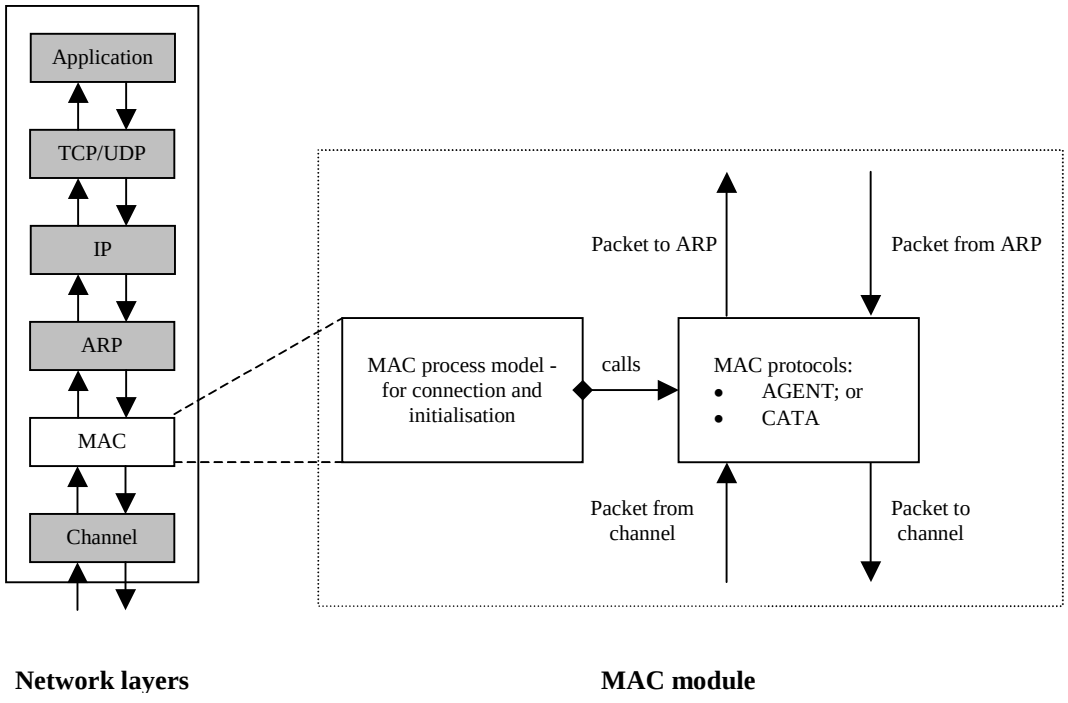

Figure 3: A structural diagram of the ad-hoc node model and the MAC module

The MAC module shown in Figure 3 shows that the generic MAC process model invokes a MAC protocol process model to control the packet flow in the medium. The purpose of having a generic MAC process model to call the MAC protocol models is to enable us to reuse part of the MAC model and to allow the users to select different protocols from the network model rather than from the node model.

A process model is used to implement all the events of a process. In this case, a MAC protocol is implemented. A process model is implemented as a state transition diagram (or a finite state machine (FSM)), which consists of states and transitions. The functions and variables of each state are programmed in C and C++. There are two types of states:

- Unforced state – It waits after entering the state until it is invoked by another process or an interrupt. It is in dark grey on this report, and red in OPNET.
- Forced state – It does not pause after entering. It flows to the next state after execution. Its colour is light grey on this report, and green in OPNET.

A MAC process model is built for general initialisations of the MAC module, and to invoke the selected MAC protocol process model. This allows us to reuse the MAC process model, the node model and the network model, and to change protocols easily.

The MAC protocol process model implements the MAC protocol. When a packet arrives from the ARP layer or the radio receiver, the protocol process model is interrupted. When transmitting, it grabs the data packet from the ARP layer, appends a header to the packet, queues it up, performs exchange of the control packets, and transmits the packet. At the receiver, when the data packet arrives at the radio receiver, the packet gets passed up to the MAC protocol process model for address checking and header removal, and forward to the ARP layer. The details regarding the protocol process models are explained in Sections 4.2 and 4.3.

To increase reusability, the parameters that are shared among all the models in this module are declared in the header file, general.h. This header file is included in all models associated with the module to avoid repeated declarations.

4.1 MAC process model

The MAC process model is implemented to:

1. Ensure that the connections to the ARP module and the transmitter/receiver module to handle the transmission of a packet are valid;
2. Assign a MAC address for each node; and
3. Invoke a MAC protocol process model to transmit and receive packets. The state diagram of the MAC process model is shown in Figure 4.

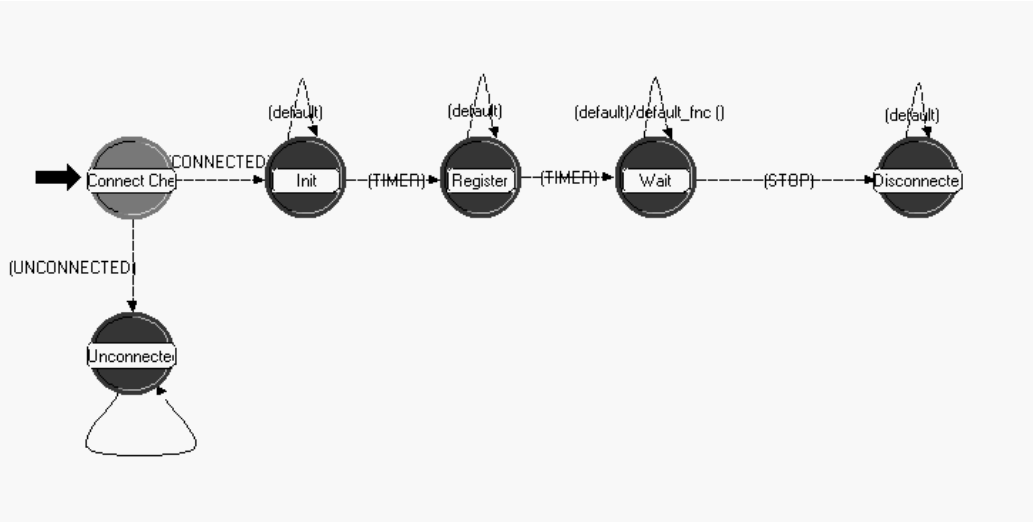

Figure 4: A diagram of an OPNET's MAC process model, which connects the MAC layer and invokes the MAC protocol process models

This process model has two model attributes:

- *MAC Address* – It defines the address for each node that will only be used in the MAC layer (default address is "Auto assigned"); and
- *Protocol Type* – It specifies the type of MAC protocol to be executed for the simulation (AGENT is the default protocol).

These values can be modified from the node's attributes in the network model, as indicated on Figure 5 below.

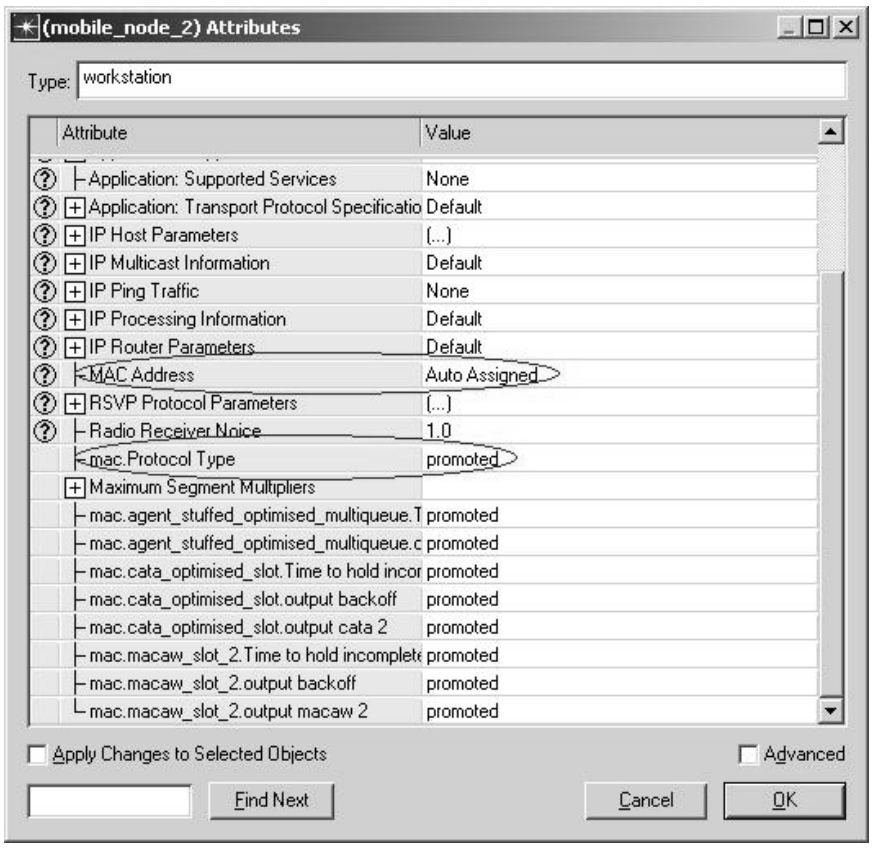

Figure 5: Attributes of a MANET node - MAC Address and Protocol Type

The advantages of being able to select the protocol type from the network model are:

- This model can be reused for any MAC protocols; and
- A new protocol can be added or a model can be renamed easily.

The steps involve in adding or changing a MAC protocol model are given in Appendix C.1.

When a node is enabled in a simulation, the MAC layer process model will be called. First, the MAC model checks if the MAC interface is connected, ensuring that the packets can be sent to the radio transmitter. It then initiates and registers its MAC address and node number. Finally, it invokes the MAC protocol process model to control the sending and receiving of packets to and from the channels, and register the ports' interrupt to notify the MAC protocol process model upon packet arrivals. After the MAC model is set up, it will remain at the *Wait* state until the node is disabled. The details about the initialisation of this model are in Appendix C.1.

4.2 Adaptive Generalized Transmission (AGENT) Protocol

AGENT is a MAC protocol designed for ad-hoc networks. This protocol is based on time allocation, where a frame is divided into multiple slots, so that each node in the network

has an assigned slot to send the packets, and each slot is divided into four Control Mini Slots (CMSs) and one data mini slot (DMS). If the node of an assigned slot has no packets to send at that time, other nodes may use the slot. This protocol can provide full use of the medium. A detail explanation of the AGENT protocol is given in [1].

As mentioned previously, AGENT uses a TDMA protocol. Thus, all the nodes in the network must be synchronised. There are two ways of doing this in the simulation environment:

1. Set an interrupt for each mini slot to keep all the nodes synchronised. However, this method would slow down the simulation due to a high number of interrupts needed to be executed; or
2. Only interrupt the model when there is data to be sent, determine the time to send a Request To Send for the data packet, and set an interrupt at that time. Thus, a time reference must be used for all the nodes in the network to synchronise the nodes.

In the initial model, the first method was used. However, due to the long simulation time required with the first method, the second method is currently used.

In [4], W.D. Blair has designed a state diagram as shown in Figure 6. Initially, this AGENT state diagram was implemented in OPNET and it was intended to be used for OPNET modelling and simulations. However, it was not appropriate for large networks or for long simulations. The time it took to run a simulation was too long when this model was used, as this model requires a large number of consistent state transitions to synchronise the mini slots for all the nodes in the network. To overcome the problem, another AGENT model was designed, as shown in Figure 7. When this model was used, there was a significant reduction in time required to run the simulations.

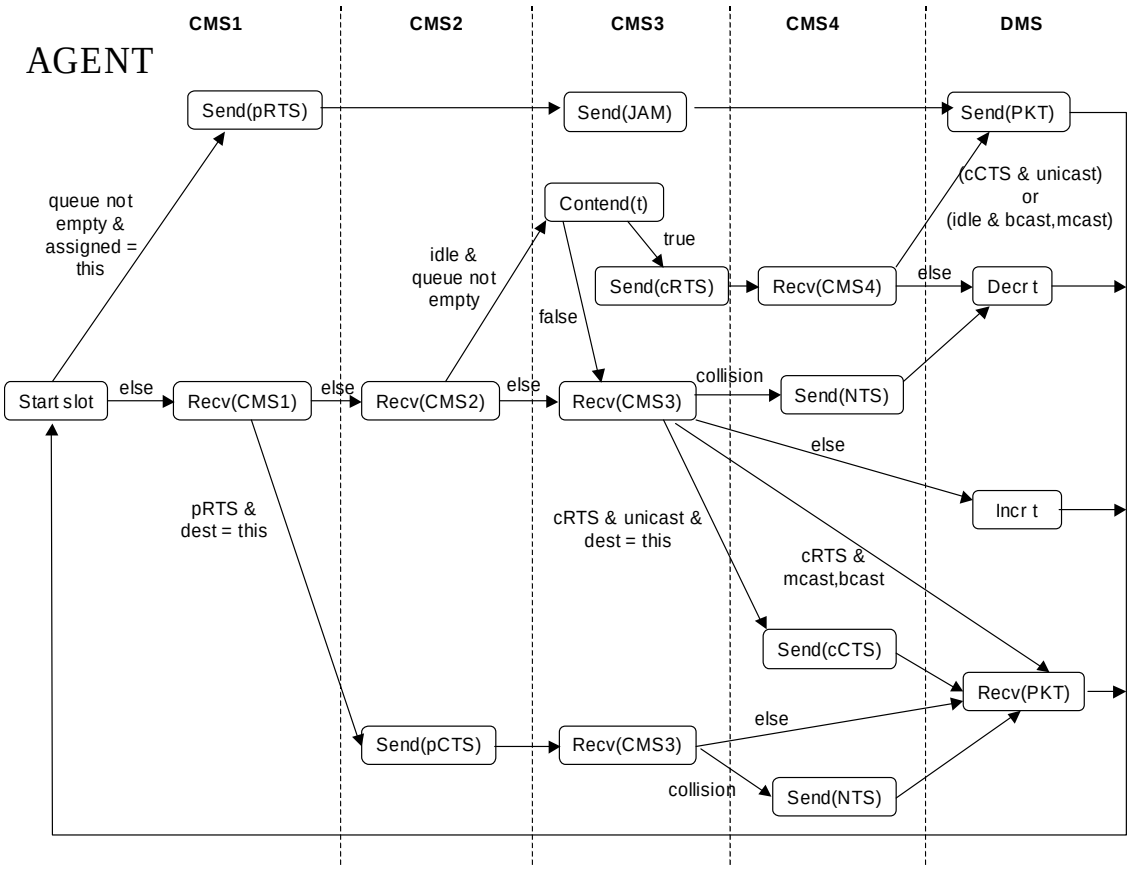

Figure 6: AGENT state diagram obtained according to Myers, A.D., Zaruba, G.V. and Syrotiuk, V.R.'s paper in [1]

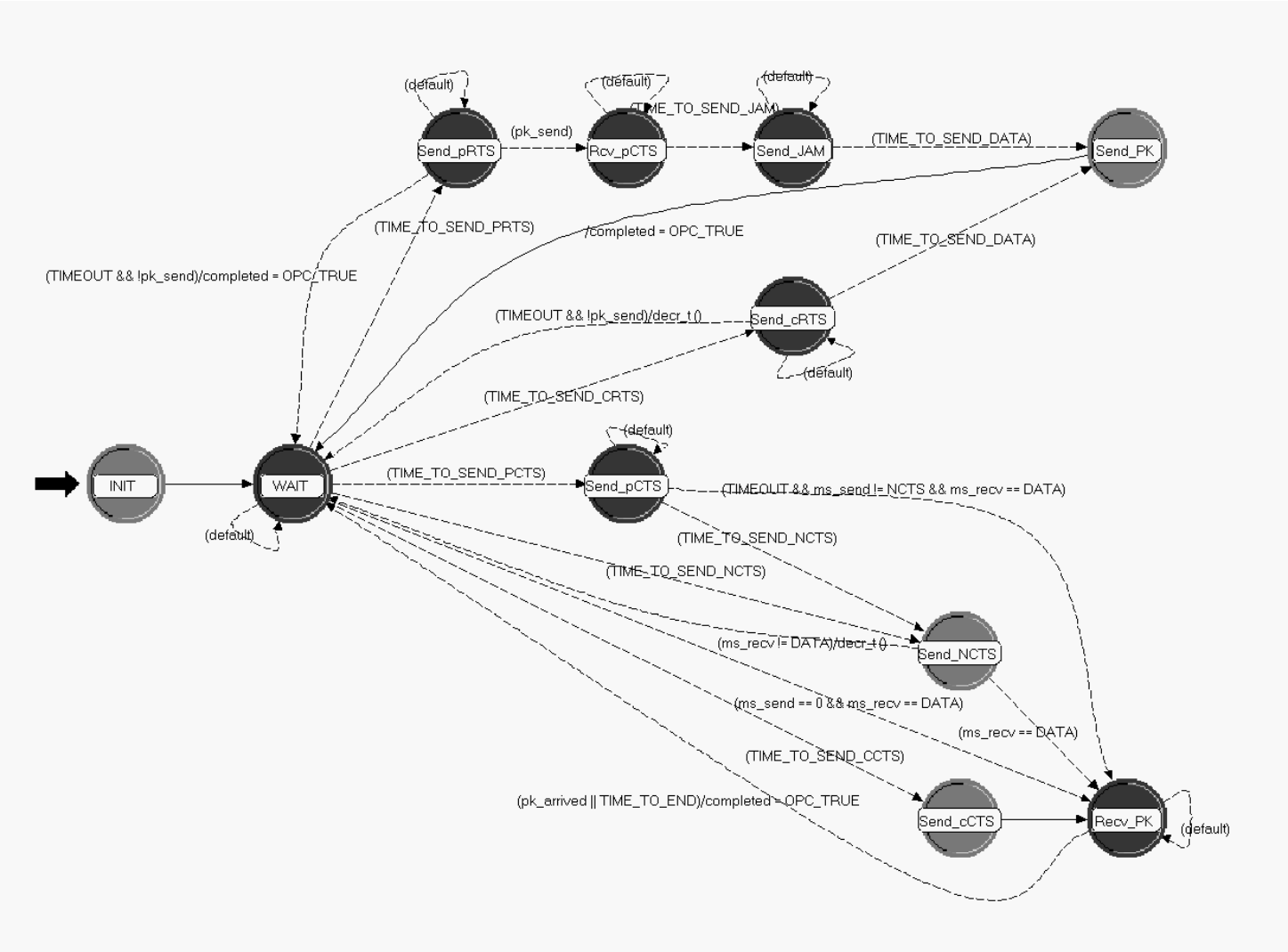

Figure 7: OPNET's process model for the AGENT protocol

Figure 7 shows a state diagram of the AGENT protocol that is currently used in OPNET simulations. It was designed based on the protocol described in [4] and the state diagram in Figure 6. The advantage of using this over the previous model is that it does not require repeated state transitions when the network is idle, as all the nodes in the network are synchronised according to the start time of the simulation.

The AGENT process model is invoked by its parent process model (i.e. the MAC layer process model) when the node is enabled. First, it obtains the parameters that are shared with its parent's process model, then it initialises the variables in the *INIT* state and flows to the *WAIT* state to wait for packets to arrive.

4.2.1 The *WAIT* state

In the WAIT state in Figure 7, the node waits for a packet to arrive either from the ARP layer or from the channel. If a packet is passed down from the ARP layer it may not necessarily be immediately queued into the MAC layer. Since AGENT is a time-slotted protocol, channel accesses occur only in a fixed periods. Consequently, it becomes very inefficient when the slots are not full. Accordingly, packets may be concatenated with several carried in a single AGENT slot. Packets arriving from the ARP layer are dropped into different queues. When the length of data in the queues is sufficient, the packets will be "stuffed" into a packet (Section 4.4 will illustrate how stuffing works in AGENT). If the assembled packet is full or near-full[3], the packet will be framed and a MAC packet will be created. An interrupt will then be set to transmit the packet at either the priority slot or the contention slot. Otherwise, the node will either wait until there is sufficient data or wait until the data can be transmitted in its priority slot.

When a packet arrives from the radio receiver in the *WAIT* state, the arriving packet will be checked as follows:

1. If the packet is destined to the current node and it is a:
   - Priority RTS, then transmit a priority CTS at the next CMS.
   - Contention RTS, then transmit a contention CTS at the next CMS if the slot is not assigned, the packet is unicasting and the function, *contend (t)*, returns TRUE, where *t* is the transmission probability (i.e. the backoff value). The details of how *t* is calculated are given in [1], and the definition of the statement, *contend (t)*, is given in Appendix E.1. Do not respond if the slot is not assigned and the packet is not unicasting, and transmit a NCTS if the slot is assigned.
2. If the packet is not destined to the current node and it is a:
   - Priority RTS, then the slot is assigned. Hence, ensure that this node and its neighbouring nodes will not transmit data at the current slot.
   - Priority CTS, then the slot will be occupied. Therefore, ensure that this node will not transmit data at the current slot.

[3] This is determined by the variable, min_contention_threshold, which is the minimum ratio of the packet size that a Data Mini Slot (DMS) would hold.

3. Otherwise, discard the packet.

If a collision occurs at the third CMS in the *WAIT* state, a timer will be set to send a NCTS packet at the next CMS. For the packet types involved in the transmissions, see Appendix D.1.

4.2.2 Priority slot

When it is time to send a priority RTS, the macro *TIME_TO_SEND_PRTS* will evaluate to TRUE, and a state transition will occur from *WAIT* to *Send_pRTS*. At the *Send_pRTS* state, a priority RTS will be sent and it will wait for a priority CTS to be received. State transition will occur from *Send_pRTS* to *Rcv_pCTS* once a priority CTS is received (for unicast) or a collision is detected (for multicast or broadcast), and that the variable, *pk_send*, is set for activating a data transmission at the data mini-slot. The module will wait at the *Rcv_pCTS* state until it is time to send JAM. Otherwise, it will return to the *WAIT* state at TIMEOUT. After sending JAM, the data will be transmitted at the data mini-slot.

4.2.3 Contention slot

As illustrated in Figure 7, a transition to the *Send_cRTS* state will occur if a contention RTS can be transmitted when there is a packet waiting to be sent, the first two (priority) mini slots were free and the *contend (t)* statement returns TRUE. Once a contention RTS is sent, it will wait for a contention CTS response, a NCTS response or no response for multicast or broadcast. If a contention CTS is received for unicast, or nothing is received for non-unicast, the data will be sent at the data mini-slot. Otherwise, it will return to the *WAIT* state.

4.3 Collision-Avoidance Time Allocation (CATA) Protocol

CATA is similar to the AGENT protocol, as it is also based on time allocation where a frame is divided into multiple slots and each slot is further divided into four CMSs and one DMS. The difference between CATA and AGENT is that CATA does not have a slot assigned permanently to each node. Thus, a node will have to get assess to a slot via the contention method. However, a priority slot will be assigned to the node that has transmitted a packet in the last slot. The CATA protocol that is modelled in OPNET is designed according to [2] and [4]. Figure 8 shows the original OPNET model for CATA.

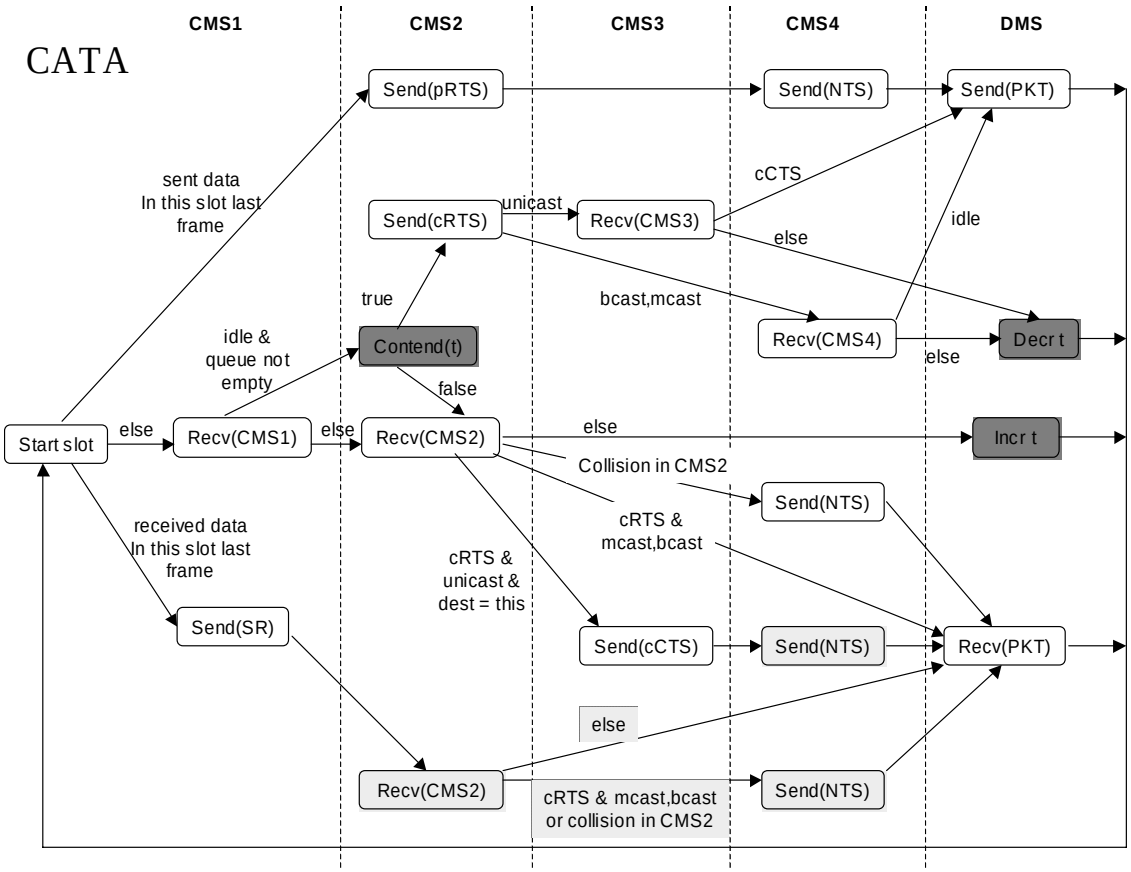

Figure 8: State diagram of CATA obtained from [2]

Similar to the AGENT protocol, the state diagram of the CATA protocol shown in Figure 8 was inappropriate for a large network or a long simulation because it was too slow to run. Hence the state diagram of CATA was redesigned to reduce the time required to run a simulation. The new CATA state diagram that is used for OPNET simulations is shown in Figure 9. A series of simulations were run confirming that there is a significant time improvement for using the model in Figure 9 over the one in Figure 8. This time improvement is due to the reduced number of state transitions required when running the simulation.

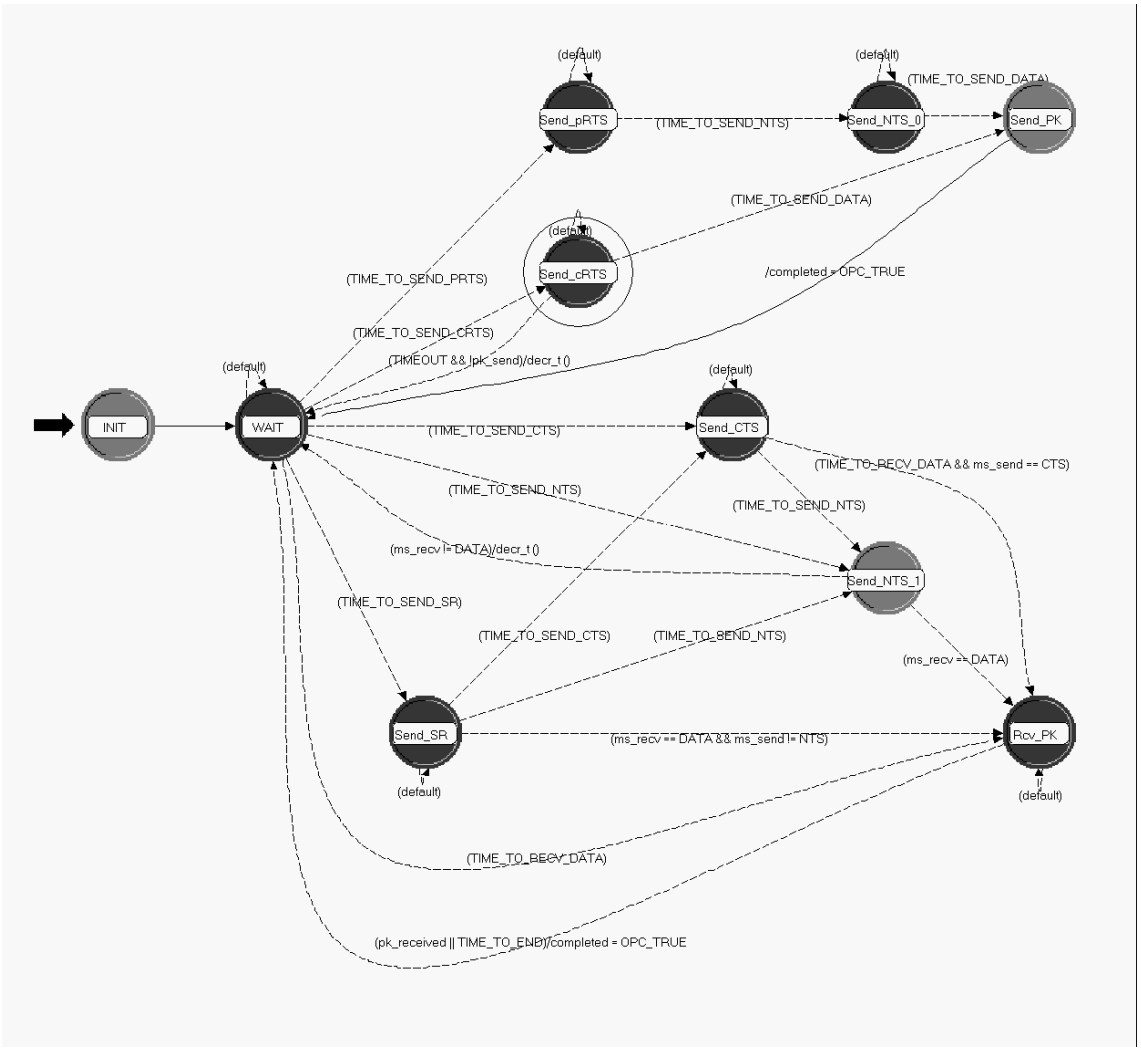

Figure 9: OPNET's process model for the CATA protocol

Once a node is enabled in a simulation, the CATA process model is invoked by its parent process model. Similar to the AGENT process model, CATA first obtains the parameters that are passed by its parent's process model. It then initialises the variables and waits for a packet to arrive either from the ARP layer or from the radio receiver. If a packet arrives from the ARP layer, it will be slotted into a queue. When there is sufficient data in the queues, the packets will be concatenating into a packet using the stuffing method explained in Section 4.4. If the assembled packet is large enough, it will be framed and an interrupt will be set for transmitting a contention packet if *contend (t)* returns TRUE, where *t* is the backoff value. Otherwise, the assembled packet is disassembled, and a timer is set to reassemble and transmit the data after the timeout.

4.3.1 The *WAIT* state

Referring to Figure 9, if a packet arrives from the radio receiver in the *WAIT* state and the slot is not assigned, the arrival packet is checked as follows:

1. If the packet is destined for the current node and it is a:
   - Priority RTS, then transmit a contention CTS if the packet is unicasting, or wait for the data to arrive if the packet is not unicasting.
   - Contention RTS, then transmit a contention CTS if the packet is unicasting and *contend (t)* returns TRUE, or wait for the data if the arrived packet is not unicasting.
2. If the packet is a SR control packet, reschedule the contention RTS interrupt if it was originally set.
3. Otherwise, discard the packet.

4.3.2 Collisions handling

If a collision occurs in:

- The first CMS, then reschedule the interrupt to send contention RTS if there is a packet waiting to be transmitted.
- The second CMS, then send a NTS packet at the forth CMS.

See Appendix D.1 for the different types of packets involved in the transmissions.

4.3.3 Sending and receiving a packet

According to Figure 9, a transition to the *Send_cRTS* state will occur if there is sufficient data in the queues, and there is no SR packet received. When it is time to send a contention CRTS and the statement, *contend (t)*, returns TRUE, the macro *TIME_TO_SEND_CRTS* will become TRUE, and a state transition will occur from *WAIT* to *Send_cRTS*. A contention RTS will then be sent. If the contention RTS is unicast, it will then wait to receive a contention CTS packet, otherwise it will wait until it is time to send data. Once a contention CTS is received and it is unicast, the variable, *pk_send*, will evaluate to TRUE so the data will be sent at the DMS. If the contention RTS is broadcast, and if a packet is received or a collision is detected in the third or forth CMS, *pk_send* will evaluate to FALSE and return to *WAIT*.

After a data packet is received at the receiver node, the receiver will transmit an SR packet to reserve the slot for the transmitter. If the transmitter has more data to send, a priority RTS will be sent at the second CMS, follow by a NTS at the fourth CMS and the data packet.

4.4 Stuffing

In the initial implementation of the MAC protocols, packets from the ARP layer were framed and transmitted regardless of how small the packet was compared to the size of the data mini-slot. This is very inefficient, as there could be a lot of space in the slot for more packet(s). Thus, to enhance the performance of the AGENT and CATA protocols (i.e.

to increase the throughput), stuffing was implemented so that the small data packets can be assembled and sent as one larger packet.

To perform stuffing, the packets from the ARP layer were slotted into the FIFO queues. Once the size of the FIFO queues are built up to a significant size, the stuffing process begins. The procedure of determining whether the stuffing process is ready to be performed is shown on the following flow diagrams (Figure 10 for AGENT and Figure 11 for CATA).

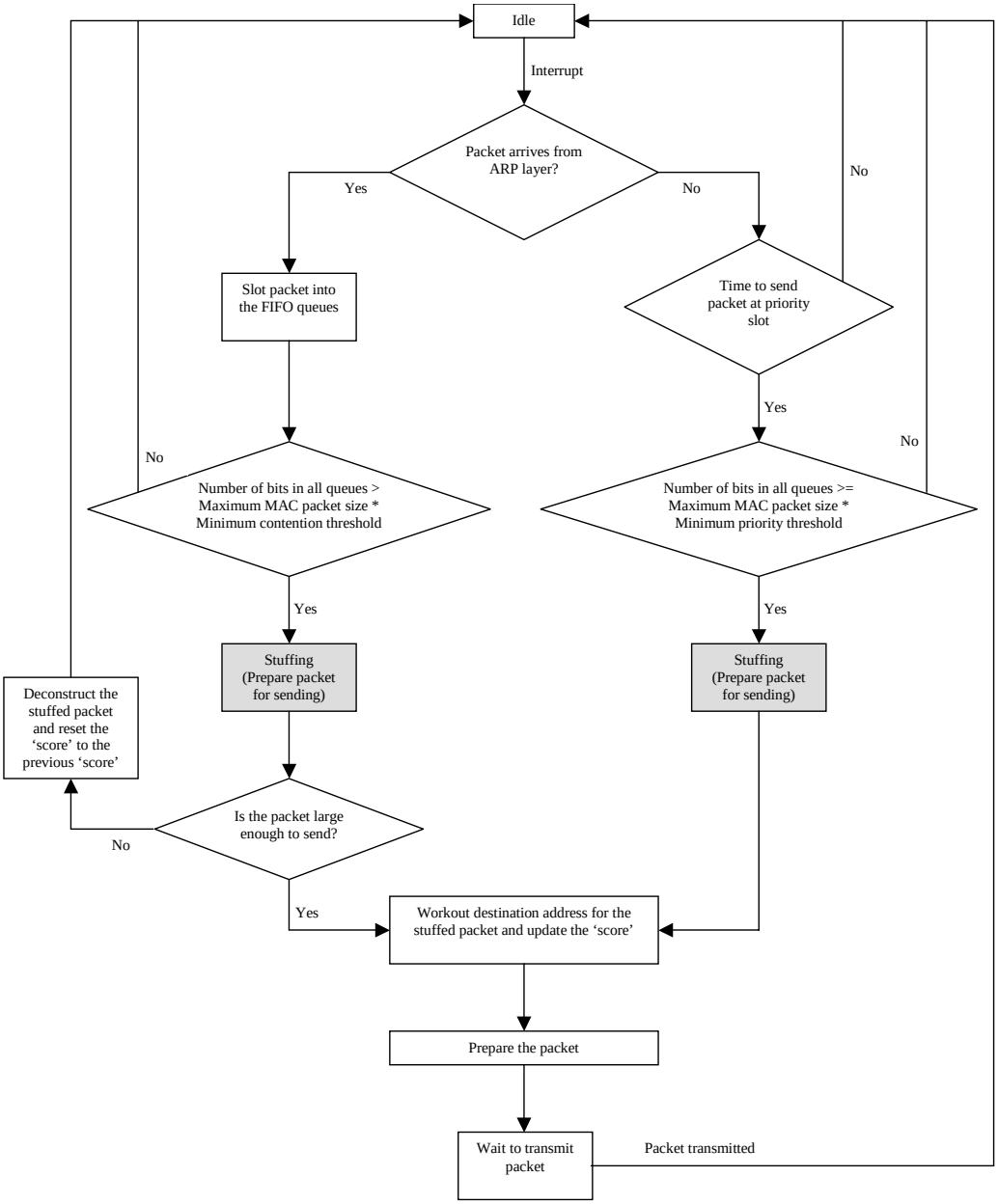

Figure 10: A flow diagram to initiate stuffing for AGENT

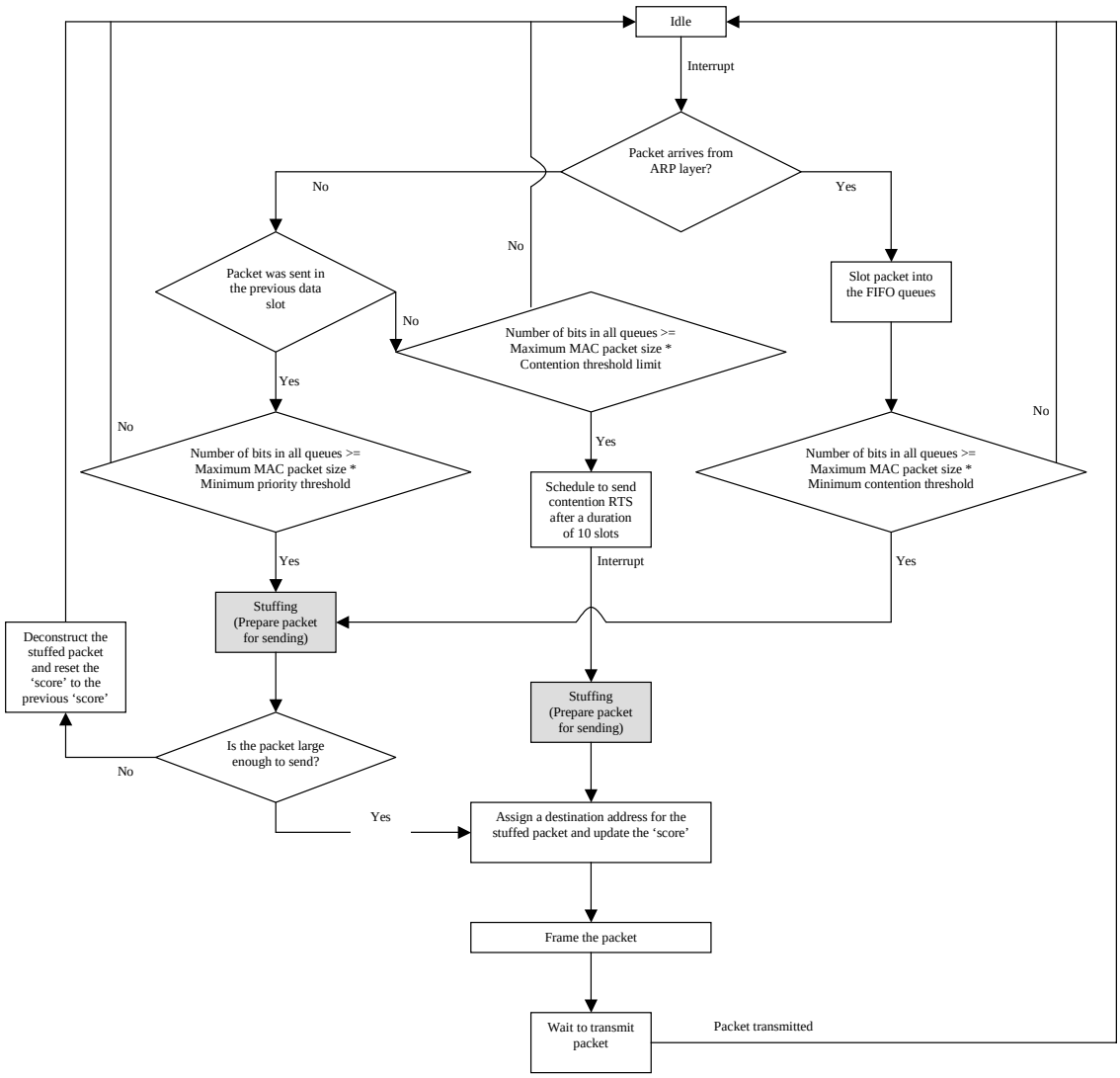

Figure 11: A flow diagram to initiate stuffing for CATA

In stuffing, each FIFO queue has an associated weight and a score that indicates the weighted bytes removed from the queue. The packet to be packed into an empty stuffing packet is selected according to the provision score calculated from the first packet of each queue. The packet selection process is then repeated to add more packets into the stuffing packet until there is insufficient room. This process is shown in the flow chart in Figure 12.

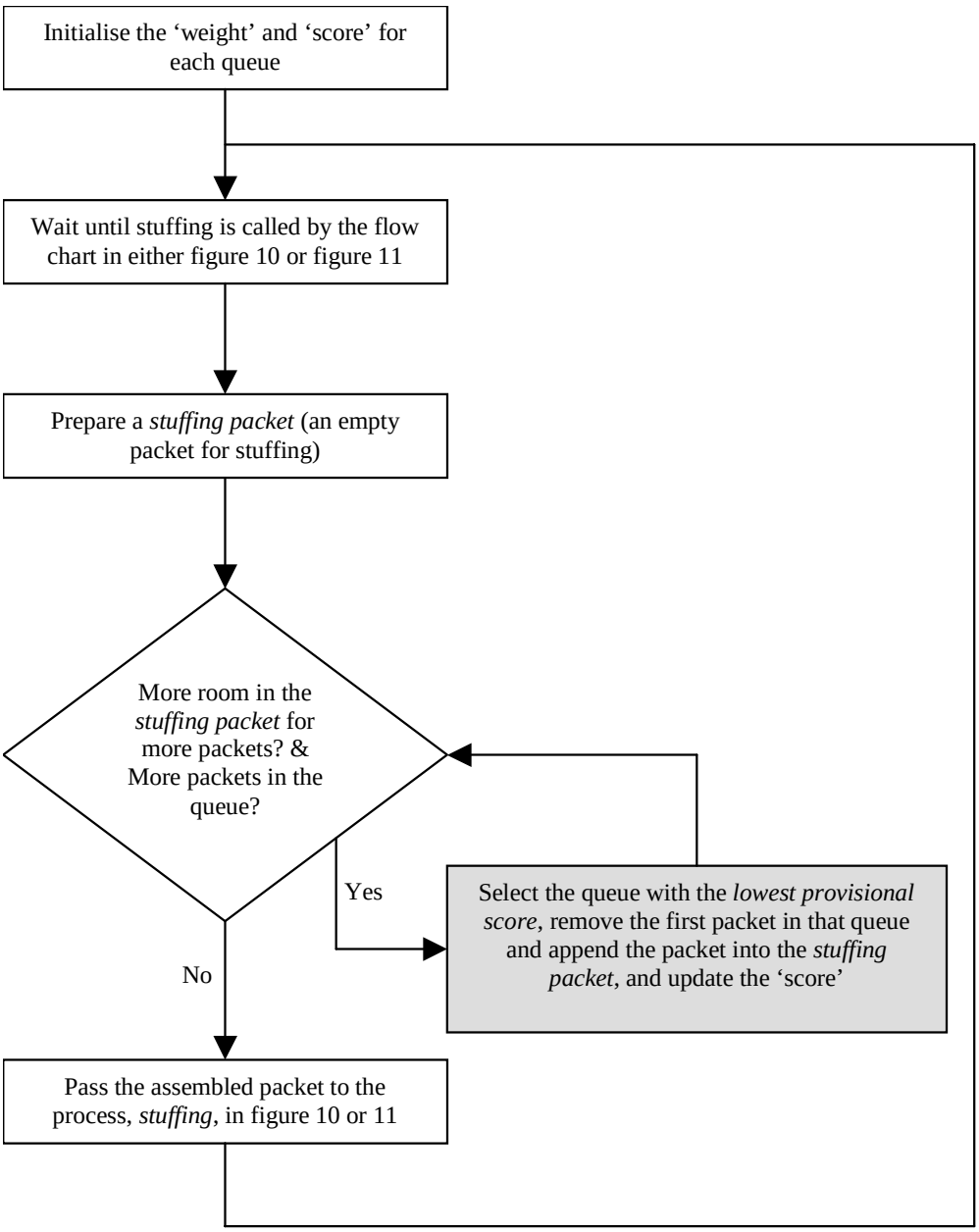

Figure 12: A flow diagram of the stuffing algorithm

In the shaded process of Figure 12, the queue with the *lowest provisional score* is obtained by calculating the *provisional score*, p, of each queue individually:

$$p = w \times l + s$$

where w is the weight assigned initially for each queue, l is the size of the first packet in the queue, and s is the score of a queue. The score, s, is equal to zero initially, and it is updated every time after a packet is stacked, with $s = p$.

Once the assembled packet is prepared and ready to be sent, the score of each queue, s, is updated by subtracting the minimum score of all non-empty queues, m. The score of each queue must have a minimum of zero.

$$s = \max < 0, s - m >$$

If the assembled packet is not to be sent, the packets in the assembled packet will be deconstructed and pushed back to the queues, and the score of each queue will be reset back to the score it had originally, before stuffing.

An example of stuffing is illustrated in Appendix F.

5. Examine the Operations of the MAC Models

The models were verified by simulating the network model described in Section 2. The simulations were executed for 15 minutes (simulation time), with the IP ping traffic starting at 2 minutes with a repeat after 10 minutes. Each time the IP ping traffic starts, five ping packets will be sent at a rate of one second per packet. The details regarding each node's settings are explained in setting 1 of Appendix A.1.

To ensure the MAC protocol models are functioning correctly, the node statistics, *Ping Requests Sent* and *Ping Replies Received*, were collected and measured. For both AGENT and CATA, we observed that the total number of *Ping Requests Sent* for each pair of nodes that are within range is the same as the total number of *Ping Replies Received*. An example of this is illustrated in Figure 13.

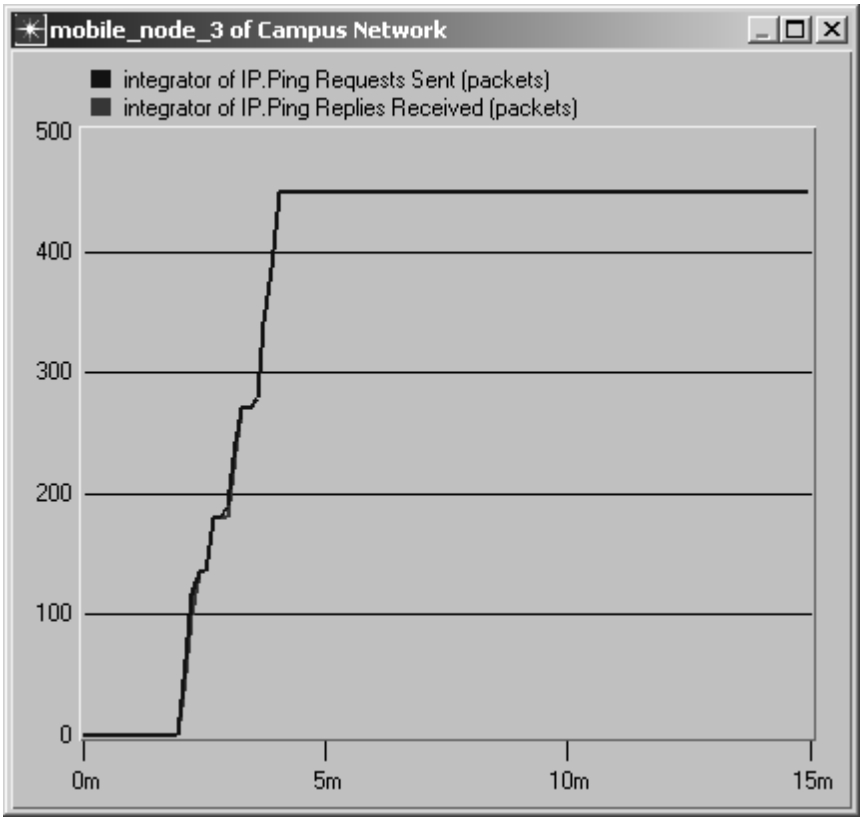

Figure 13: This graph shows the number of ping requests sent from node 3 to node 6 and the number of ping replies received by node 3 from node 6. It indicates that the total number packets sent is the same as the total number of packets received.

The graph above indicates that the IP ping packets were received properly, and hence the protocol models are working. However, this cannot demonstrate the functions of the protocols in detail. The functions of a protocol were verified by assessing the processes of the model in an execution of the OPNET simulation, using the log printed with simulation time and stages of the model stated. An example of a section of the log file for AGENT model simulation is given below (Log 1). It indicates that a collision can be avoided when the hidden node is attempting to send a packet to a non-hidden node. Since node 4 is hidden from node 6, it cannot hear the priority CTS reply from node 6, so it sends a contention RTS to node 0. However, node 0 detected a collision, as it is not hidden from node 6. Thus it broadcast a NCTS. See Appendix G for more log samples for AGENT and CATA.

Log 1: This shows that Node 1 wants to send a packet to Node 6 while Node 4 wants to send a packet to Node 0. However, the hidden node, Node 4, is recognised when Node 0 has detected a collision at the third CMS and sends a NCTS packet, and that a priority packet is successfully transmitted from Node 1 to Node 6. Figure 24 shows the processes that the log illustrates. Note that the control number represents the type of a control packet (see Appendix G for the representation of each value).

```
# 120.028500: CONTROL sent (Source: 1, Destination: 6), control 1, nm 1,
ID 263
120.029262: Node 6 received priority RTS (need to send pCTS)
120.029269: Node 4 heard priority RTS
120.029292: Node 5 heard priority RTS
# 120.029625: CONTROL sent (Source: 6, Destination: 1), control 2, nm 5,
ID 284
120.029861: Node 5 heard priority CTS - reschedule cRTS
120.030387: Node 1 received priority CTS (will send packet)
120.030394: Node 3 heard priority CTS - reschedule cRTS
# 120.030750: CONTROL sent (Source: 1, Destination: 6), control 3, nm 2,
ID 305
120.030750: Node 4 (mac_contend())
# 120.030750: CONTROL sent (Source: 4, Destination: 0), control 4, nm 2,
ID 306
120.031224: Node 0 detected collision at CMS3 (need to send NCTS)
# 120.031875: CONTROL sent (Source: 0, Destination: 0), control 6, nm 5,
ID 347
120.032383: Node 4 received NCTS [unicast] (cannot send packet)
120.032644: Destination address = 0
# 120.033000: DATA sent (Source: 1, Destination: 6), nm 3, ID 61
120.041012: Node 6 receive data
```

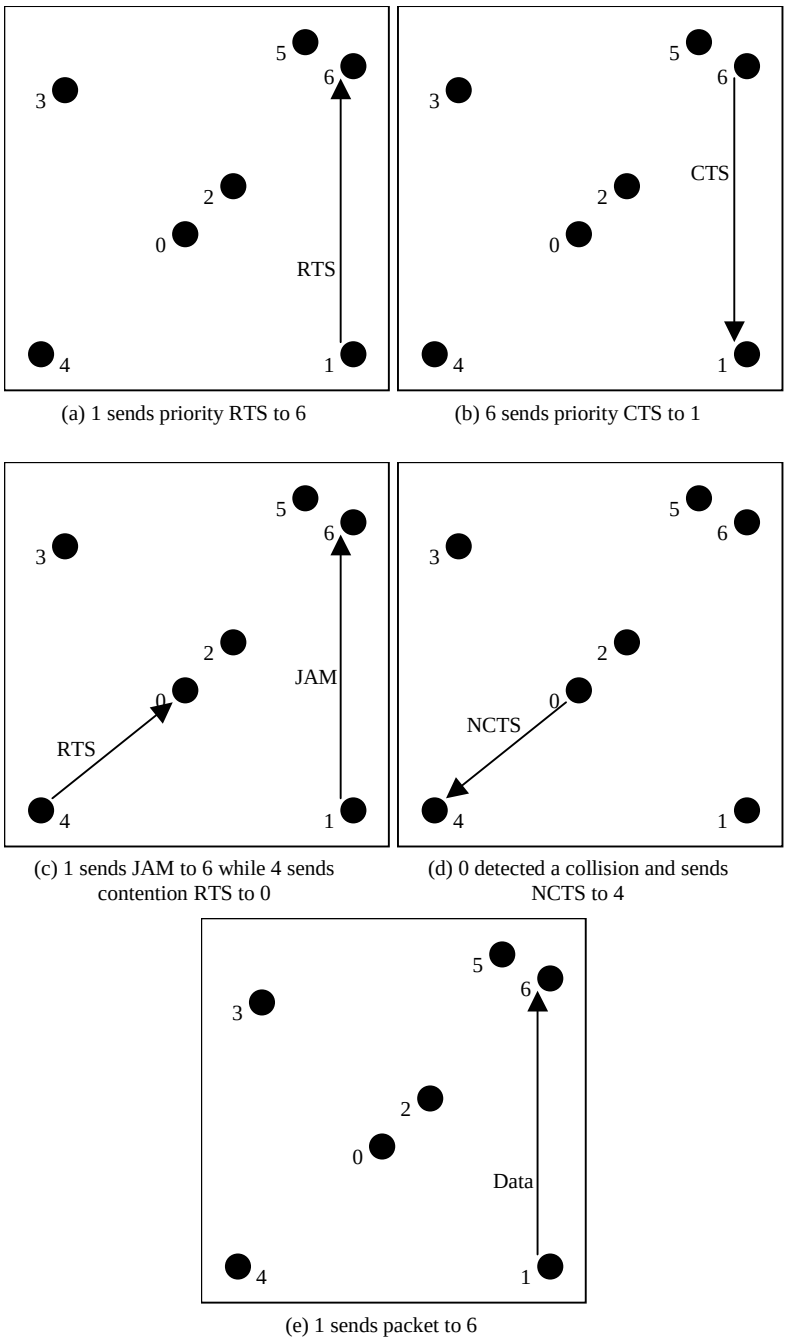

Figure 14: A graphical view illustrating the processes described in Log 1.

After tracing through a series of log files for AGENT and CATA with different node settings (see Appendix A.1 for the node settings), the models were confirmed to be operating correctly.

The average ping response times calculated for both protocols are given in

Table 1. These indicate that AGENT is more responsive than CATA. This is because AGENT uses a combination of contention method and TDMA scheme where there is a dedicated time slot for each node to guarantee a transmission, whereas with CATA each node can only gain access to the medium through contention. Thus, with high network load and high node connectivity, the performance of CATA decreases. Due to the high network load used in the simulations, high ping response times were observed for CATA.

Table 1: The average ping response time for AGENT and CATA

	AGENT	CATA
Average ping response time (sec)	0.35	0.5

6. Conclusion

We have successfully implemented AGENT and CATA in OPNET. This report has shown how they are modelled, integrated, verified and compared. Our initial testing has demonstrated that AGENT performs better than CATA. This is because AGENT has an allocated time slot for each node and it also uses the contention method, where other nodes can use the unused allocated slot. However, a case when AGENT may perform less as well in comparison to CATA is when only a few nodes are trying to send lots of packets while many other nodes do not have any packets to send.

In the future, we will be looking at implementing other MAC protocols such as the Medium Access Protocol for Wireless LANs (MACAW). These models will be used to carry out network simulations using the Headline 2000 data to compare and analyse the performance of different MAC protocols in a military tactical setting.

7. Acknowledgments

I would like to thank Bill Blair and Michael Carter for their contributions to the stuffing model. Michael Carter has also helped with designing the improved version of the AGENT model and implementing the stuffing for AGENT. I would also like to thank Ian Grivell for his help on some modelling ideas.

8. References

[1] Myers, A.D., Zaruba, G.V. and Syrotiuk, V.R. (2002) *An Adaptive Generalized Transmission Protocol for Ad Hoc Networks*, ACM Journal of Mobile Networks and Applications 7, 493-520, 6 December 2002.

[2] Tang, Zhenyu and Garcia-Luna-Aceves J.J. (1999) *A protocol for topology-dependent transmission scheduling in wireless networks*, Proc. IEEE WCNC'99, Vol. 3, 1333-1337, September 1999.

[3] Carter, M. (2004) *A Discrete Event Simulation Framework for Radio Ad Hoc Networks*, DSTO-TN-xxxx, 2004.

[4] Blair, W.D. (2003) *Potential Shortcomings of Selected Media Access Control Protocols For Wireless Ad Hoc Networks*, Journal of Battlefield Technology, Vol. 6, Issue 4.

Appendix A : Network Models

A.1. Node settings

Verifying the models via simulation is essential for ensuring that the AGENT and CATA protocols are working correctly and that process models are bug free. The same network model and settings have been used to simulate and check different MAC models. Three different settings were deployed for simulating the MAC models.

Setting 1: General

Node	Attribute	Value
IP Ping	IP ping parameters -> row 0 -> packet size	436
	IP ping parameters -> row 1 -> packet size	436
Mobile_node_0	IP ping traffic -> row 0 -> • start time • IP address • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 255.255.255.255 600 constant unlimited
Mobile_node_1	IP ping traffic -> row 0 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_3 600 constant unlimited
	IP ping traffic -> row 1 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_4 600 constant unlimited
	IP ping traffic -> row 2 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_5 600 constant unlimited

	IP ping traffic -> row 3 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_6 600 constant unlimited
Mobile_node_3	IP ping traffic -> row 0 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_0 600 constant unlimited
	IP ping traffic -> row 1 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_4 600 constant unlimited
	IP ping traffic -> row 2 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_5 600 constant unlimited
	IP ping traffic -> row 3 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_6 600 constant unlimited
Mobile_node_4	IP ping traffic -> row 0 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_0 600 constant unlimited
	IP ping traffic -> row 1 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_2 600 constant unlimited

Mobile_node_5	IP ping traffic -> row 0 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_0 600 constant unlimited
	IP ping traffic -> row 1 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_1 600 constant unlimited
	IP ping traffic -> row 2 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_2 600 constant unlimited
	IP ping traffic -> row 3 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_3 600 constant unlimited
Mobile_node_6	IP ping traffic -> row 0 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_0 600 constant unlimited
	IP ping traffic -> row 1 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_1 600 constant unlimited
	IP ping traffic -> row 2 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_2 600 constant unlimited

	IP ping traffic -> row 3 -> • start time • host name • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 mobile_node_3 600 constant unlimited

Setting 2: Random

Same as the table above (setting 1), but change all the inter-repetition time (sec) to 5 seconds, inter-repetition time (PDF) to exponential and maximum repetition count to 10.

Setting 3: Broadcast

Use the same setting as setting 2, but add the following to all the mobile nodes.

Node	Attribute	Value
Mobile_nodes	IP host parameters -> interface information -> multicast mode	enable
	IP ping traffic -> row *x* -> • start time • IP address • inter-repetition time (sec) • inter-repetition time (PDF) • maximum repetition count	 120 255.255.255.255 5 exponential 10

Appendix B : Node Model

B.1. Processor Settings

Processor	Attribute	Value
rt_0	Channel->data rate Channel->packet formats Channel->bandwidth Channel->frequency	512,000 All formatted, unformatted 512 10
rr_0	Channel->data rate Channel->packet formats Channel->bandwidth Channel->frequency	512,000 All formatted, unformatted 512 10

B.2. Packet Stream and Statistic Wire Settings

Packet Stream / Statistic Wire	Attribute	Value
arp->mac	Src stream Dest stream	Src stream [0] Dest stream [0]
mac->arp	Src stream Dest stream	Src stream [0] Dest stream [1]
mac->rt_0	Src stream Dest stream	Src stream [1] Dest stream [0]
rr_0->mac	Src stream Dest stream	Src stream [0] Dest stream [1]
rr_0->mac (Statistic wire)	Src stat Rising edge trigger	Radio receiver.collision status enable

Appendix C : Process Models

C.1. MAC process model

C.1.1 Interfaces

Attribute	Type	Default Value	Description
MAC Address	Integer	Auto assigned	Storing the node's own MAC address
Protocol Type	Integer	AGENT	Chooses the protocol it will use

C.1.2 Initialisation and calling the protocol process model

Initialising the MAC module involves:

- Ensuring that it has a valid transmission channel;
- Assigning a MAC address for the node; and
- Starting a MAC protocol process model.

C.1.3 Modifications

3. To rename a protocol process model:

If you have renamed a protocol process model that is already exists in the MAC process model, you need to include that process model's name of that protocol into the MAC process model. To do this, you are required to:

- Include that new process model by going to the *Declare child processes*; and
- Change the *mac_protocol_process_model* in
  #*define* <*PROTOCOL*>*_PROCESS_NAME mac_protocol_process_model*.

4. To create a new protocol process model:

If you have implemented a new protocol process model, but it is not initialised in the MAC process model, you need to initialise it in the MAC process model by doing the following:

- Include the new model by going to the *Declare child processes*;
- Add a new value, PROTOCOL_VALUE, for the new protocol under *Protocol Type* in the *Model attributes*;
- Add #*define* <*PROTOCOL*>*_PROCESS_NAME mac_protocol_model*;
- Add #*define* <*PROTOCOL*> *(protocol_type* == <*PROTOCOL_VALUE*>*)*;
- Declare a structure for passing parameters to the protocol process model in the header file, general.h;
- Add the required code to initialise the structure that needs to be passed to the protocol process model; and
- Add code to create the protocol process model and register the ports that associates to it.

  The ports that need to be registered are two packet stream ports, IN_STRM_FROM_ARP and IN_STRM_FROM_NTWK, and one statistic line.

C.2. AGENT

C.2.1 Interfaces required

Attribute	Type	Default Value	Description
Time to hold incomplete packet arrivals	Double	15 sec	Used for stuffing. It specifies the time it keeps an incomplete arrived packet before dropping it.
Minimum Priority Threshold	Double	0	A ratio of the DMS size. It is the minimum size of the existing packets in the queues that must be meet to assemble a priority packet.
Minimum Contention Threshold	Double	1	A ratio of the DMS size. It is the minimum amount of data (as a ratio of a fully packed size) in the queues that must be waiting before a contention packet will be assembled. If the assembled packet is smaller than the Minimum Contention Packet Threshold, the packet will be disassembled and the data pushed back onto the waiting queues.
Minimum Contention Packet Threshold	Double	0.5	A ratio of the DMS size. It represents the smallest assembled packet size to be sent at the contention slot. If the packet is smaller, then the packet will be disassembled and the data pushed back onto the waiting queues.

C.3. CATA

C.3.1 Interfaces required

Attribute	Type	Default Value	Description
Time to hold incomplete packet arrivals	Double	15 sec	Used for stuffing. It specifies the time it keeps an incomplete arrived packet before dropping it.
Minimum Priority Threshold	Double	0.8	A ratio of the DMS size. It is the minimum size of the packets in the queues that must be met to assemble a packet to be sent at a priority slot. If the assembled packet is smaller than the Minimum Packet Threshold, the packet will be disassembled and the data pushed back onto the waiting queues.
Minimum Contention Threshold	Double	1	A ratio of the DMS size. It is the size limit of the packets in the queues that must be met to assemble a contention packet. If the assembled packet is smaller than the Minimum Packet Threshold, the packet will be disassembled and the data pushed back onto the waiting queues.
Minimum Packet Threshold	Double	0.8	A ratio of the DMS size.
Contention Threshold Limit	Double	0.5	A ratio of the DMS size. It is the size limit of the packets in the queues that must be met to trigger a contention interrupt after a set period of time.

Appendix D : Packet Formats

The packet formats below are designed for the purpose running OPNET simulations. They are not real packet formats. Thus the fields in a packet do not necessarily require bits allocation, i.e. a field can have zero bytes. The fields with zero bytes are for sending data or information for use in OPNET simulations only.

D.1. AGENT and CATA

AGENT and CATA protocols both used the same packet types and the same packet formats – control and data.

D.1.1 Control

Control packets are used in the CMSs. Each control packet consists of a source address, a destination address, a packet id number, its control type (i.e. RTS, CTS, NCTS, NTS and SR) and the number of packets sent by the node. Figure 15 shows the format of a control packet.

Figure 15: AGENT's and CATA's control packet

D.1.2 Data

Data packet can only be sent in a DMS. It consists of a source address, a destination address, the data from the higher layer, the type of the packet and the number of packets sent by the node. Figure 16 shows the format of a data packet. The size of a data packet cannot be greater than 512 bytes.

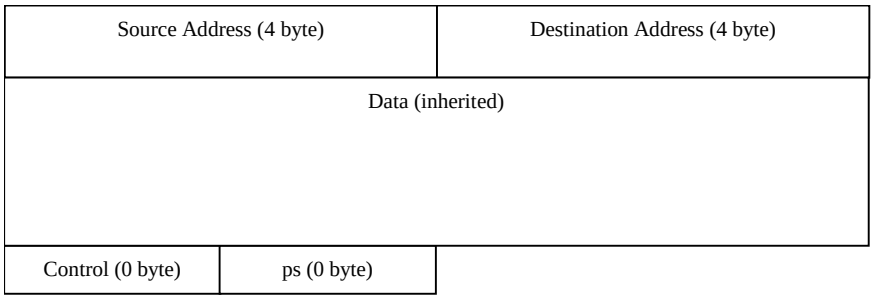

Figure 16: AGENT's and CATA's data packet

Appendix E : Backoff Algorithms

E.1. contend (t)

contend (t) is a function that returns TRUE if *t* is greater than the uniform distributed random number (between 0 and 1), where *t* is the transmission probability. The code for this function is given below:

```
Boolean contend (t)
    {
    float rand_num = (float) rand()/(float) RAND_MAX;
    return ((Boolean) (rand_num <= t));
    }
```

E.2. AGENT

t is increased by a factor of 2 when:

- The network is idle.

t is decreased by a factor of 2 when:

- There is a collision detected at the contention RTS mini-slot;
- A contention CTS is not received successfully (if it is a unicast); or
- The contention NCTS mini-slot is not idle (if it is a multicast or broadcast).

E.3. CATA

t is increased by a factor of 2 when:

- The network is idle.

t is decreased by a factor of 2 when:

- A contention CTS is not received (if unicast); or
- The medium is not idle at the fourth control mini-slot (if multicast or broadcast).

Appendix F : An Example of Stuffing

Assume that the packet size for transmission is 1000 bytes. If there are three queues in the implementation, the packets arriving from the ARP layer can be distributed into each of these queues randomly. Let the weights of queue 1, 2 and 3 be 5, 10 and 20 respectively, and the content of the queues illustrated in Figure 17 below.

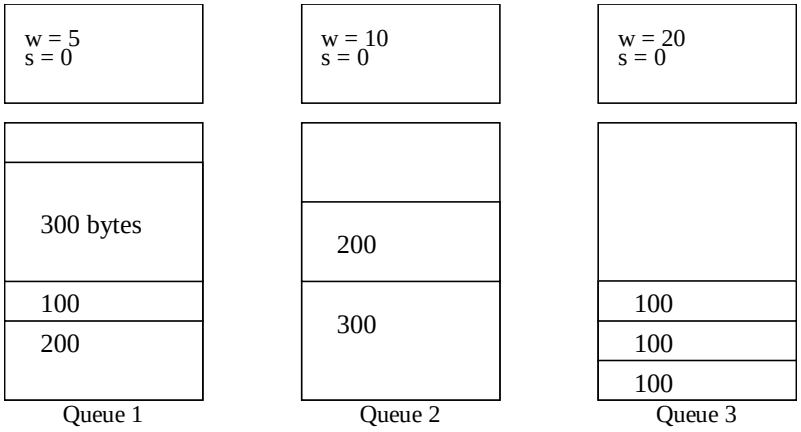

Figure 17: The weight, the initial score and the size of the packets in each queue before stuffing begins.

Figure 18 to Figure 23 will show how stuffing works in the simulation.

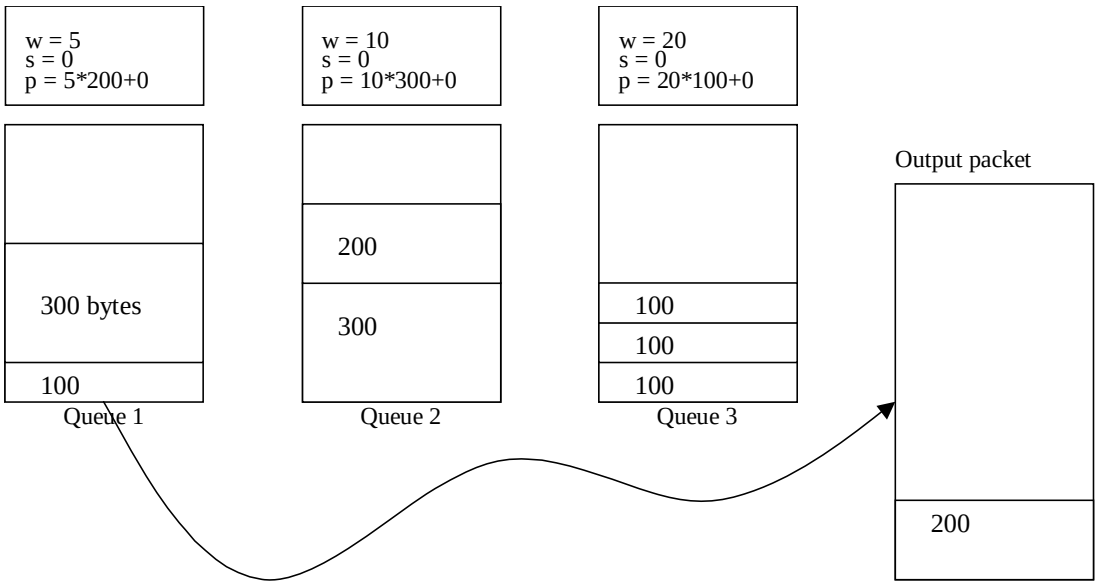

Figure 18: Queue 1 has the minimum provisional score; therefore, its first packet gets stuffed into the output buffer.

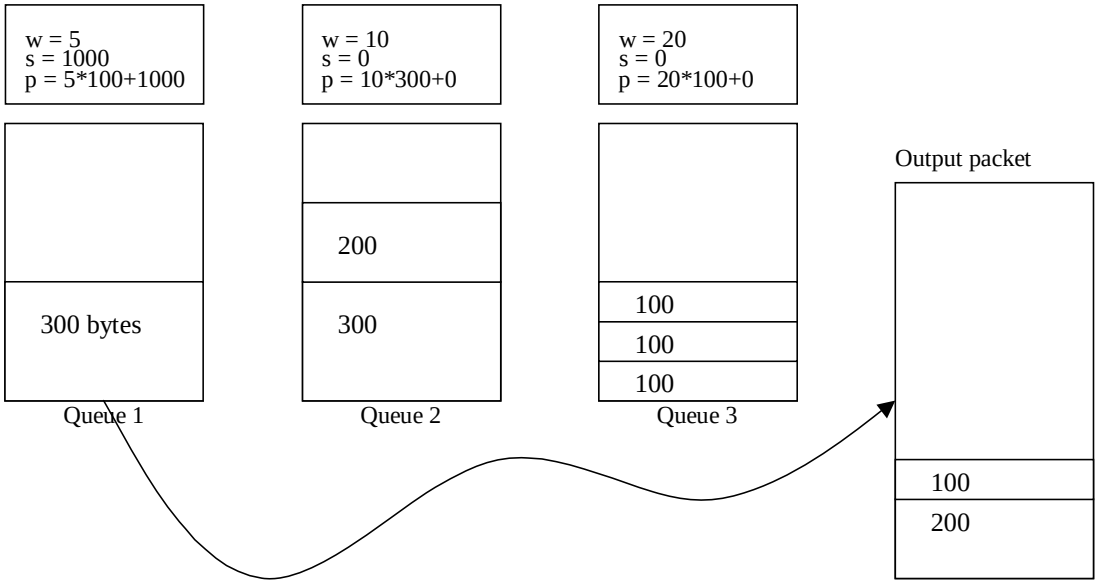

Figure 19: The score in queue 1 gets updated, and its new provisional score is calculated. The new provisional score shows that queue 1 is still the winner. Therefore, its next packet gets stuffed into the output packet.

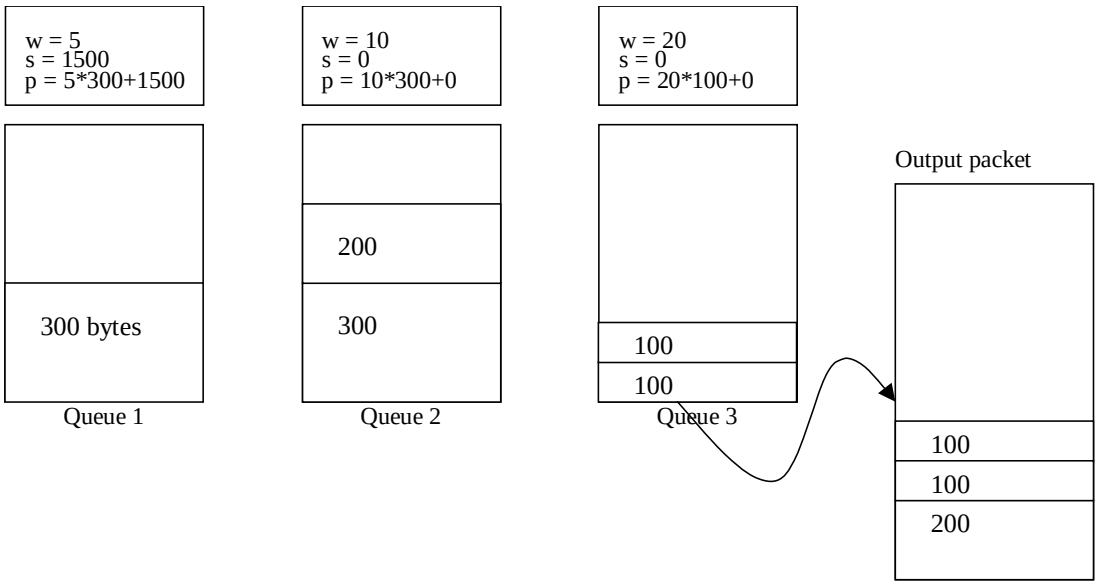

Figure 20: The score and provisional score are updated. This time, queue 3 is the winner.

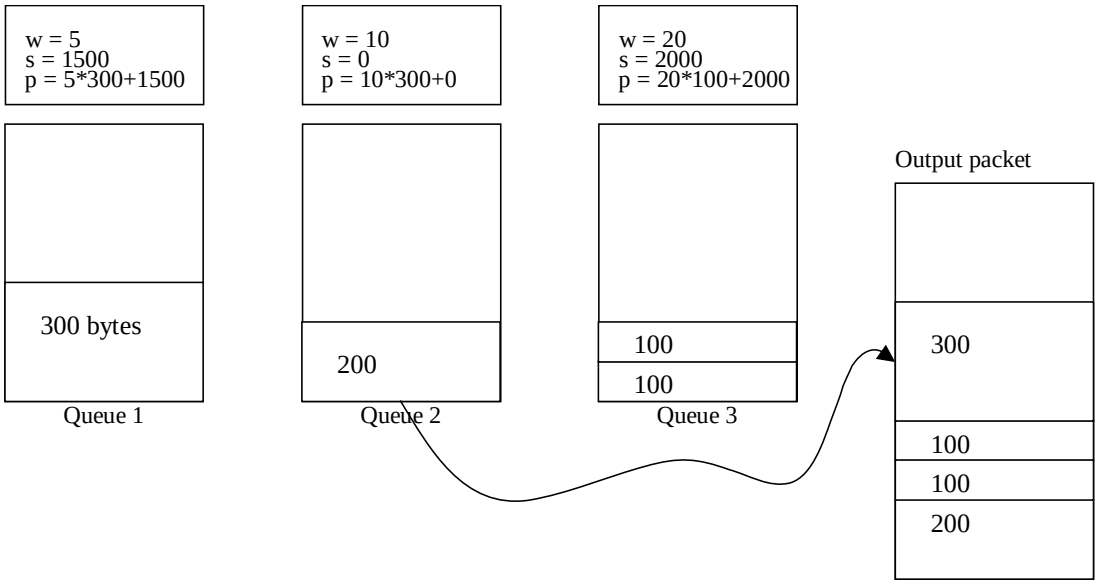

Figure 21: The provisional score, p, of queue 1 and queue 2 are the same, thus, either of these packets can be stuffed into the output packet.

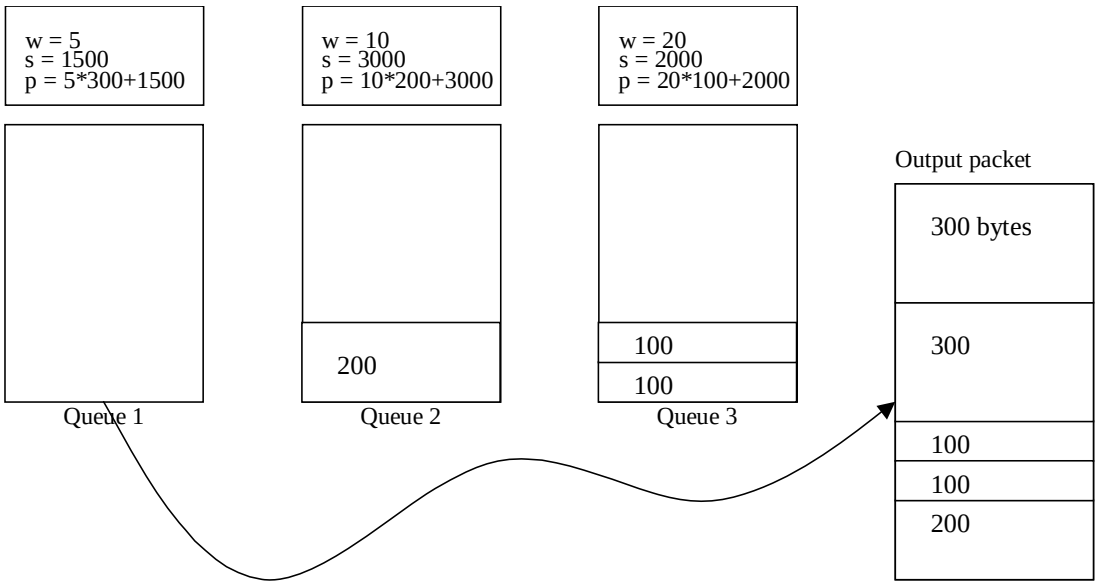

Figure 22: The packet from queue 1 filled the output packet.

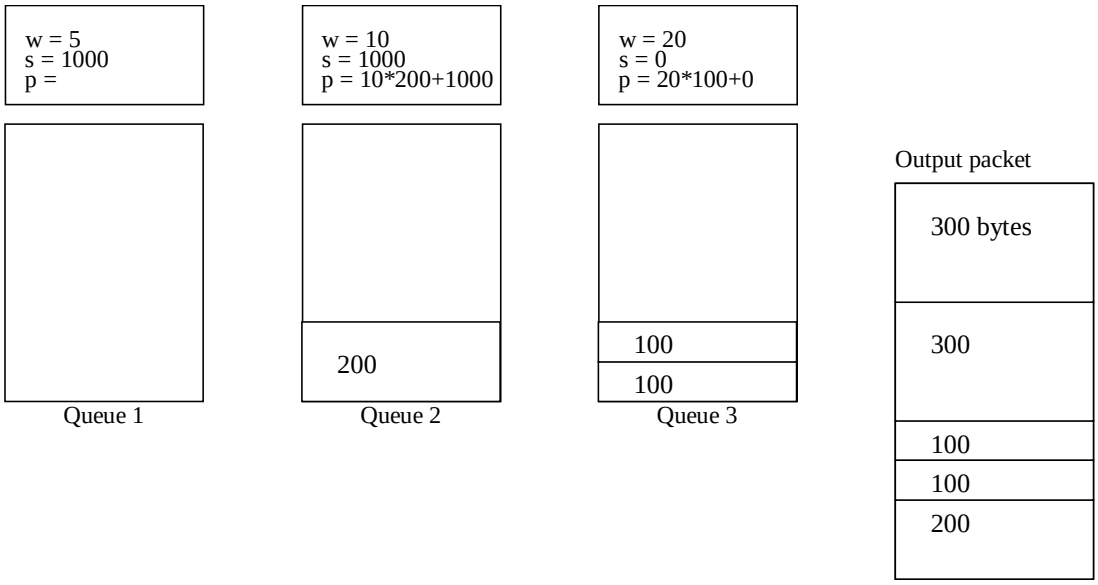

Figure 23: When the output packet is full, it will be queued for sending, and the score will become either zero or score minus 2000, as 2000 is the lowest score out of queue 2 and queue 3 (the non-empty queues).

The stuffing process repeats until all the queues are empty.

Appendix G : Simulation Log

To understanding the outputs from the model simulations, we need to understand the symbols and the numeric values in the log file. These are explained below:

- The '#' represents a node is sending a packet at that time.
- The number at the beginning of the line is the current simulation time.
- Source is the sender.
- Destination is the receiver.
- nm is the number of packet sent.
- ID is the packet id.
- The following are related to AGENT only:
  - control 1 is priority RTS.
  - control 2 is priority CTS.
  - control 3 is JAM.
  - control 4 is contention RTS.
  - control 5 is contention CTS.
  - control 6 is NCTS.
- The following are related to CATA only:
  - control 1 is SR.
  - control 2 is priority RTS.
  - control 3 is contention RTS.
  - control 4 is CTS.
  - control 5 is NTS.

G.1. AGENT

Below we present a few important sections of the log files from the AGENT simulations. Log 2, Log 3 and Log 4 show some successful transmissions.

Log 2: Node 6 has the priority slot and has a packet to send to node 0, therefore it sends a priority RTS to node 0. Node 0 is not hidden from any nodes, therefore, no contention RTS is sent and the packet is transmitted successfully. This log is illustrated graphically in Figure 24.

```
# 120.001500: CONTROL sent (Source: 6, Destination: 0), control 1, nm 1,
ID 70
120.001736: Node 5 heard priority RTS
120.002208: Node 0 received priority RTS (need to send pCTS)
120.002262: Node 1 heard priority RTS
120.002269: Node 3 heard priority RTS
# 120.002625: CONTROL sent (Source: 0, Destination: 6), control 2, nm 1,
ID 91
120.003133: Node 4 heard priority CTS - reschedule cRTS
120.003169: Node 3 heard priority CTS - reschedule cRTS
120.003224: Node 1 heard priority CTS - reschedule cRTS
120.003278: Node 5 heard priority CTS - reschedule cRTS
120.003333: Node 6 received priority CTS (will send packet)
```

```
# 120.003750: CONTROL sent (Source: 6, Destination: 0), control 3, nm 2,
ID 112
# 120.006000: DATA sent (Source: 6, Destination: 0), nm 3, ID 69
120.013958: Node 0 receive data
120.013978: Packet of size 3712 has arrived at node 0
```

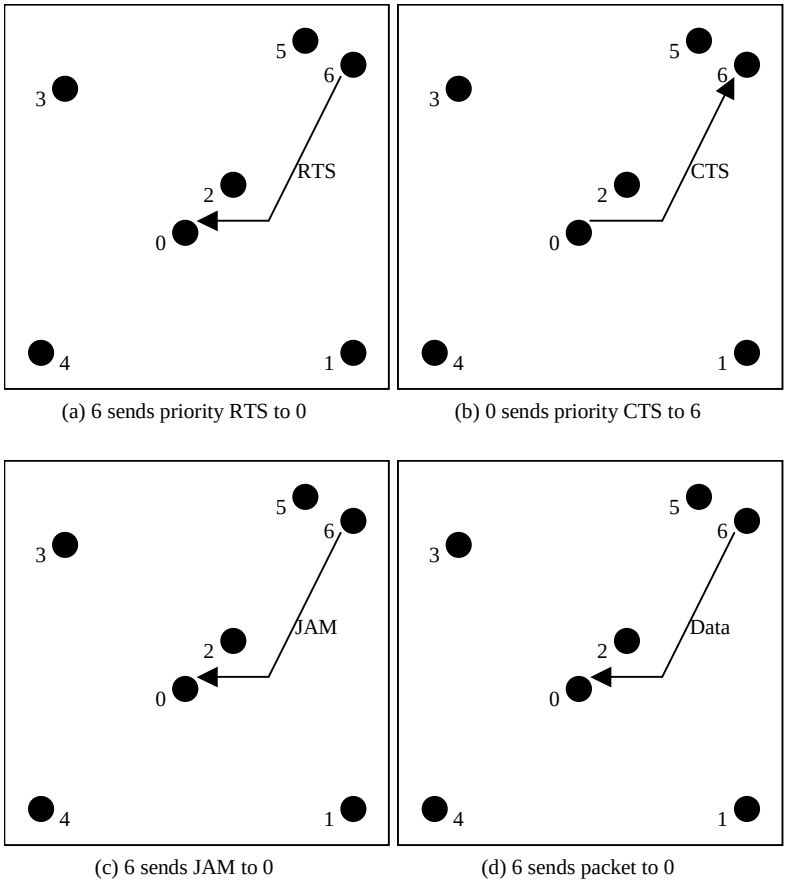

Figure 24: Graphical representation of Log 2.

Log 3: Node 0 is having its priority slot, and it needs to send a packet to node 6. First, node 0 sends a priority RTS to node 6. Node 6 replies with a priority CTS and sends a JAM. While node 6 is sending a JAM, node 4 sends a contention RTS to node 0 as it could not hear the priority CTS packet from node 6. However, no nodes would send a NCTS packet to node 4, because node 0 will be sending a packet and node 6 cannot hear node 4. Since node 4 is not receiving a CTS packet, it will not send the data packet. Hence, node 6 is able to receive data successfully. See Figure 25 for a clearer understanding of this.

```
# 120.015000: CONTROL sent (Source: 0, Destination: 6), control 1, nm 2,
ID 159
120.015508: Node 4 heard priority RTS
120.015544: Node 3 heard priority RTS
120.015599: Node 1 heard priority RTS
120.015653: Node 5 heard priority RTS
```

```
120.015708: Node 6 received priority RTS (need to send pCTS)
# 120.016125: CONTROL sent (Source: 6, Destination: 0), control 2, nm 4,
ID 180
120.016361: Node 5 heard priority CTS - reschedule cRTS
120.016833: Node 0 received priority CTS (will send packet)
120.016887: Node 1 heard priority CTS - reschedule cRTS
120.016894: Node 3 heard priority CTS - reschedule cRTS
# 120.017250: CONTROL sent (Source: 0, Destination: 6), control 3, nm 3,
ID 201
120.017250: Node 4 (mac_contend())
# 120.017250: CONTROL sent (Source: 4, Destination: 0), control 4, nm 1,
ID 222
120.018883: Destination address = 0
# 120.019500: DATA sent (Source: 0, Destination: 6), nm 4, ID 158
120.027458: Node 6 receive data
```

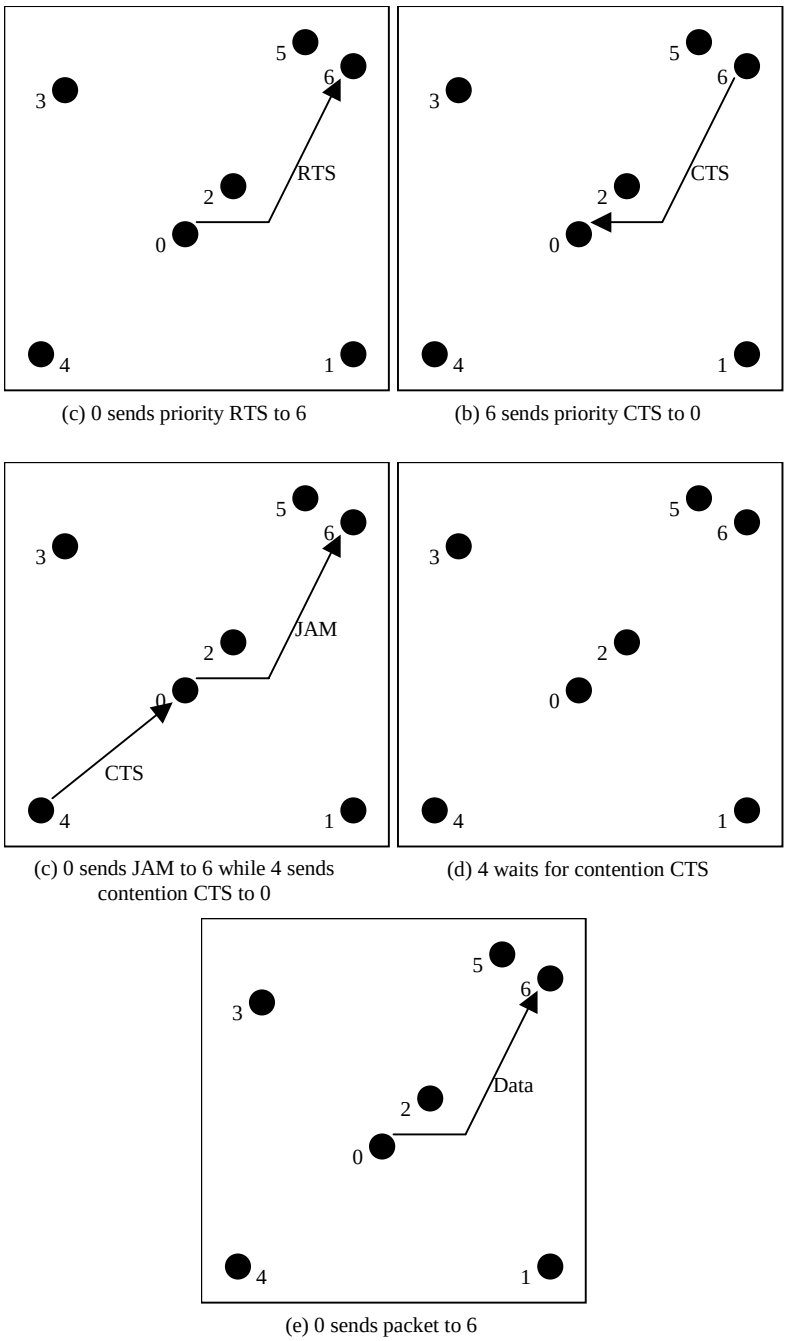

Figure 25: Representation of Log 3.

Log 4: An example of a successful contention transmission occur when there is no priority transmission occurring, and only one node has the authority to send a contention RTS.

```
120.138750: Node 3 (mac_contend())
# 120.138750: CONTROL sent (Source: 3, Destination: 6), control 4, nm 6,
ID 1042
```

```
120.138750: Node 5 (mac_contend())
120.139519: Node 6 received contention RTS (need to send cCTS)
# 120.139875: CONTROL sent (Source: 6, Destination: 3), control 5, nm
11, ID 1063
120.140644: Node 3 received contention CTS [unicast] (will send packet)
# 120.141000: DATA sent (Source: 3, Destination: 6), nm 7, ID 867
120.149019: Node 6 receive data
```

However, there are cases where unsuccessful transmissions occur when there is no priority transmission detected. This can be shown in Log 5.

Log 5: This log indicates that there were three nodes trying to send a contention packet at the same slot: node 5 to node 0; node 3 to node 6; and node 6 to node 3. However, because all the contention RTS packets collide, no data packets got sent.

```
# 120.044250: CONTROL sent (Source: 5, Destination: 0), control 4, nm 1,
ID 394
120.044250: Node 3 (mac_contend())
# 120.044250: CONTROL sent (Source: 3, Destination: 6), control 4, nm 1,
ID 395
120.044250: Node 4 (mac_contend())
120.044250: Node 6 (mac_contend())
# 120.044250: CONTROL sent (Source: 6, Destination: 3), control 4, nm 6,
ID 396
120.044567: Node 2 detected collision at CMS3 (need to send NCTS)
120.044682: Node 2 detected collision at CMS3 (need to send NCTS)
120.044778: Node 0 detected collision at CMS3 (need to send NCTS)
120.044833: Node 0 detected collision at CMS3 (need to send NCTS)
120.044917: Node 1 detected collision at CMS3 (need to send NCTS)
# 120.045375: CONTROL sent (Source: 2, Destination: 2), control 6, nm 1,
ID 457
# 120.045375: CONTROL sent (Source: 0, Destination: 0), control 6, nm 6,
ID 458
# 120.045375: CONTROL sent (Source: 1, Destination: 1), control 6, nm 4,
ID 459
120.045764: Node 5 received NCTS [unicast] (cannot send packet)
120.045807: Node 3 detected collision (cannot send packet)
120.045817: Node 6 received NCTS [unicast] (cannot send packet)
120.046012: Node 6 detected collision (cannot send packet)
120.046028: Node 5 received NCTS [unicast] (cannot send packet)
120.046167: Node 5 received NCTS [unicast] (cannot send packet)
```

G.2. CATA

Below we present a few important logs from the CATA simulations. Log 6 shows an unsuccessful transmission, and successful transmissions are illustrated in Log 7, Log 8 and Log 9.

Log 6: Initially, the backoff is low, so all the nodes in the network would try to send as a contention. Hence, unsuccessful transmission is observed – nodes 1, 3, 4, 5 and 6 sent a contention RTS, but collision was detected.

```
120.002625: Node 1 (mac_contend())
# 120.002625: CONTROL sent (Source: 1, Destination: 0), control 3, nm 1,
ID 67
120.002625: Node 3 (mac_contend())
# 120.002625: CONTROL sent (Source: 3, Destination: 0), control 3, nm 1,
ID 68
120.002625: Node 4 (mac_contend())
# 120.002625: CONTROL sent (Source: 4, Destination: 0), control 3, nm 1,
ID 69
120.002625: Node 5 (mac_contend())
# 120.002625: CONTROL sent (Source: 5, Destination: 0), control 3, nm 1,
ID 70
120.002625: Node 6 (mac_contend())
# 120.002625: CONTROL sent (Source: 6, Destination: 1), control 3, nm 1,
ID 71
120.002736: Node 6 detected collision in CMS3 (cannot send packet)
120.002736: Node 5 detected collision in CMS3 (cannot send packet)
120.002942: Node 2 detected collision at CMS2 (need to send NTS)
120.003044: Node 0 detected collision at CMS2 (need to send NTS)
120.003269: Node 1 detected collision in CMS3 (cannot send packet)
120.003269: Node 3 detected collision in CMS3 (cannot send packet)
120.003289: Node 4 detected collision in CMS3 (cannot send packet)
# 120.004875: CONTROL sent (Source: 2, Destination: -1), control 5, nm
1, ID 172
*** No data in queues - node 2
# 120.004875: CONTROL sent (Source: 0, Destination: -1), control 5, nm
1, ID 173
120.004875: Node 0 set a timer to send a CRTS at 120.151125
```

After a few iterations, the backoff is adjusted. Therefore, node 4 is able send a packet to node 0 and node 4 obtained a priority transmission for the next slot (see Log 7).

Log 7: Node 4 sent a contention RTS to node 0. After node 0 received the RTS packet, it replied to node 4 with a CTS. Node 4 received the CTS packet, and thus it sends the data packet. Once node 0 received the data packet, it schedules itself to send a SR packet to node 4. After receiving the SR packet, node 4 sends a priority CTS to node 2, because it has nothing for node 0. Node 2 replies node 4 with a CTS, transmits a NTS and waits for the data. At the DMS, node 4 sends data to node 2. At the following slot, node 2 sends a SR packet. However, node 4 has nothing to send, therefore, no transmission occurred.

```
120.043125: Node 4 (mac_contend())
# 120.043125: CONTROL sent (Source: 4, Destination: 0), control 3, nm 3,
ID 466
120.043125: Node 3 (mac_contend())
120.043125: Node 1 (mac_contend())
120.043125: Node 5 (mac_contend())
120.043125: Node 6 (mac_contend())
120.043633: Node 0 received contention RTS (unicast - need to send CTS)
```

```
# 120.044250: CONTROL sent (Source: 0, Destination: 4), control 4, nm 4,
ID 487
120.044758: Node 4 received CTS [unicast] (will send packet)
# 120.045375: CONTROL sent (Source: 0, Destination: 4), control 5, nm 5,
ID 508
# 120.046500: DATA sent (Source: 4, Destination: 0), nm 4, ID 62
120.046500: Preparing a packet at node 4... 3712 - destined to 2
120.054258: Node 0 received data
120.054258: Node 0 data received - send SR
120.054278: Packet of size 3712 has arrived at node 0
120.054278: Preparing a packet at node 0... 3712 - destined to -1
120.054278: slot_time_passed = 0.012278
120.054278: Node 0 schedule self interrupt for CRTS (at 120.056625,
queue length = 7424)
# 120.055500: CONTROL sent (Source: 0, Destination: 4), control 1, nm 6,
ID 557
120.055891: SR packet arrived at Node 2 (Do nothing)
120.056008: SR packet arrived at Node 4 (send pRTS)
120.056044: SR packet arrived at Node 3 (reschedule cRTS)
120.056099: SR packet arrived at Node 1 (reschedule cRTS)
120.056153: SR packet arrived at Node 5 (reschedule cRTS)
120.056208: SR packet arrived at Node 6 (reschedule cRTS)
# 120.056625: CONTROL sent (Source: 4, Destination: 2), control 2, nm 5,
ID 578
120.057399: Node 2 received priority RTS (unicast - need to send CTS)
# 120.057750: CONTROL sent (Source: 2, Destination: 4), control 4, nm 4,
ID 599
# 120.058875: CONTROL sent (Source: 4, Destination: 2), control 5, nm 6,
ID 620
# 120.058875: CONTROL sent (Source: 2, Destination: 4), control 5, nm 5,
ID 621
# 120.060000: DATA sent (Source: 4, Destination: 2), nm 7, ID 530
*** No data in queues - node 4
120.068024: Node 2 received data
120.068024: Node 2 data received - send SR
*** No data in queues - node 2
120.068044: Packet of size 3712 has arrived at node 2
120.068044: slot_time_passed = 0.012544
120.068044: Node 2 schedule self interrupt for CRTS (at 120.205125)
120.069000: Node 0 self interrupt (did not receive data)
120.069000: Destination address = -1
# 120.069000: CONTROL sent (Source: 2, Destination: 4), control 1, nm 6,
ID 686
120.069389: SR packet arrived at Node 5 (reschedule cRTS)
120.069391: SR packet arrived at Node 0 (reschedule cRTS)
120.069442: SR packet arrived at Node 6 (reschedule cRTS)
120.069557: SR packet arrived at Node 3 (reschedule cRTS)
120.069614: SR packet arrived at Node 1 (reschedule cRTS)
120.069774: SR packet arrived at Node 4 (Do nothing)
120.082500: Node 2 self interrupt (did not receive data)
120.082500: Node 2 set a timer to send a CRTS at 120.218625
```

The following log is an example of a possible simultaneous transmission when the nodes are hidden from one another.

Log 8: This shows that node 6 has received data from node 1 successfully while node 3 has also received data from node 4. This is because node 6 is hidden from node 4, and node 3 is hidden from node 1, thus, simultaneous transmission is feasible even though collision is detected by node 2.

```
# 120.393000: CONTROL sent (Source: 5, Destination: 1), control 1, nm
21, ID 3623
120.393236: SR packet arrived at Node 6 (reschedule cRTS)
120.393389: SR packet arrived at Node 2 (reschedule cRTS)
120.393653: SR packet arrived at Node 0 (reschedule cRTS)
120.393663: SR packet arrived at Node 3 (Do nothing)
120.393792: SR packet arrived at Node 1 (send pRTS)
# 120.394125: CONTROL sent (Source: 1, Destination: 6), control 2, nm
26, ID 3644
120.394125: Node 4 (mac_contend())
# 120.394125: CONTROL sent (Source: 4, Destination: 3), control 3, nm
21, ID 3645
120.394599: Node 0 detected collision at CMS2 (need to send NTS)
120.394887: Node 6 received priority RTS (unicast - need to send CTS)
120.394914: Node 3 received contention RTS (unicast - need to send CTS)
# 120.395250: CONTROL sent (Source: 6, Destination: 1), control 4, nm
27, ID 3686
# 120.395250: CONTROL sent (Source: 3, Destination: 4), control 4, nm
30, ID 3687
120.395682: Node 2 detected collision at CMS3 (Something's wrong?
Collisions at CMS3!!)
120.396039: Node 4 received CTS [unicast] (will send packet)
# 120.396375: CONTROL sent (Source: 1, Destination: 6), control 5, nm
27, ID 3728
# 120.396375: CONTROL sent (Source: 0, Destination: -1), control 5, nm
28, ID 3749
120.396375: Destination address = -1
# 120.396375: CONTROL sent (Source: 6, Destination: 1), control 5, nm
28, ID 3750
# 120.396375: CONTROL sent (Source: 3, Destination: 4), control 5, nm
31, ID 3751
# 120.397500: DATA sent (Source: 4, Destination: 3), nm 22, ID 3453
120.397500: Preparing a packet at node 4... 3712 - destined to 1
# 120.397500: DATA sent (Source: 1, Destination: 6), nm 28, ID 3602
120.397500: Preparing a packet at node 1... 3712 - destined to 5
120.405512: Node 6 received data
120.405512: Node 6 data received - send SR
120.405539: Node 3 received data
120.405539: Node 3 data received - send SR
120.406500: Node 5 self interrupt (did not receive data)
120.406500: Node 5 set a timer to send a CRTS at 120.542625
```

Although, simultaneous transmission is feasible in some hidden node cases, collisions could occur in a lot of other cases. These must be avoided by detecting multiple CTS or a collision in the third CMS by the sender. The log below illustrates the detection of a hidden node.

Log 9: Node 6 is trying to send a packet to node 1 while node 2 is trying to send to node 4. However, node 4 is hidden from node 6, but node 2 is visible to all the nodes. Thus, the data packet from node 6 to node 1 would collide with the packet from node 2 to node 4, so node 1 is unable to receive its' packet.

```
120.448125: Node 6 (mac_contend())
120.448125: Node 2 (mac_contend())
120.448125: Node 0 (mac_contend())
120.461625: Node 6 (mac_contend())
# 120.461625: CONTROL sent (Source: 6, Destination: 1), control 3, nm
31, ID 4236
120.461625: Node 2 (mac_contend())
# 120.461625: CONTROL sent (Source: 2, Destination: 4), control 3, nm
21, ID 4237
120.461625: Node 0 (mac_contend())
120.462387: Node 1 received contention RTS (unicast - need to send CTS)
120.462399: Node 4 received contention RTS (unicast - need to send CTS)
# 120.462750: CONTROL sent (Source: 1, Destination: 6), control 4, nm
33, ID 4278
# 120.462750: CONTROL sent (Source: 4, Destination: 2), control 4, nm
27, ID 4279
120.463224:  Node 0 detected collision at CMS3 (Something's wrong?
Collisions at CMS3!!)
120.463364: Node 2 received multiple CTSs - collision in CMS3 [unicast]
(cannot send packet)
120.463512: Node 6 received CTS [unicast] (will send packet)
# 120.463875: CONTROL sent (Source: 1, Destination: 6), control 5, nm
34, ID 4320
# 120.463875: CONTROL sent (Source: 4, Destination: 2), control 5, nm
28, ID 4321
# 120.465000: DATA sent (Source: 6, Destination: 1), nm 32, ID 3320
120.465000: Preparing a packet at node 6... 3712 - destined to 3
120.465000: Destination address = 4
120.473012: Node 1 received data
120.473012: Node 1 data received - send SR
*** No data in queues - node 1
120.474000: Node 4 self interrupt (did not receive data)
*** No data in queues - node 4
```

DISTRIBUTION LIST

Modelling of Medium Access Control (MAC) Protocols for Mobile Ad-hoc Networks

R. Chau

AUSTRALIA

DEFENCE ORGANISATION

Task Sponsor
Director General Integrated Capability Development

S&T Program
Chief Defence Scientist
FAS Science Policy
AS Science Corporate Management
} shared copy
Director General Science Policy Development
Counsellor Defence Science, London (Doc Data Sheet)
Counsellor Defence Science, Washington (Doc Data Sheet)
Scientific Adviser to MRDC Thailand (Doc Data Sheet)
Scientific Adviser Joint
Navy Scientific Adviser (Doc Data Sheet and distribution list only)
Scientific Adviser - Army (Doc Data Sheet and distribution list only)
Air Force Scientific Adviser (Doc Data Sheet and distribution list only)
Scientific Adviser to the DMO M&A (Doc Data Sheet and distribution list only)
Scientific Adviser to the DMO ELL

Information Sciences Laboratory
Chief of Information Networks Division (Doc Data Sheet and Distribution List Only)
Research Leader Military Communications
Head Mobile Networks Group
Head Wireless Systems Group
Head Network Management Group
Dr P.A. Blackmore
M. Hue
I. Grivell
R. Chau

DSTO Library and Archives
Library Edinburgh 2 copies
Australian Archives

Capability Development Group
Director General Maritime Development (Doc Data Sheet only)
Director General Land Development
Director Information Infrastructure Development
Deputy Director Information Networks (Ms. Tina Ormsby)
Deputy Director Mobile Communications (Cmdr I. McConachie)
Deputy Director Long Range Communications (Lt. Col. K Toohey)

SO Mobile Communications – Land (Maj. A. Dillon)
SO Mobile Communications – Maritime

Chief Information Officer Group
Head Information Systems Division (Doc Data Sheet only)
Director General Information Services (Doc Data Sheet only)
Senior Manager Network Operations
AS Information Strategies and Futures (Doc Data Sheet only)
Director General Simulation Office (Doc Data Sheet only)

Strategy Group
Director General Military Strategy (Doc Data Sheet only)
Assistant Secretary Governace and Counter-Proliferation

Navy
Director Navy C4ISREW Systems (DNC4ISREW)
Deputy Director Navy C4 Networks
Director General Navy Capability, Performance and Plans, Navy Headquarters (Doc Data Sheet only)
Director General Navy Strategic Policy and Futures, Navy Headquarters (Doc Data Sheet only)
SO (Science & Technology) – Maritime Development, Russell Offices, Canberra, ACT (Doc Data sheet & Exec Summ)
Maritime Operational Analysis Centre, Deputy Director (Operations) and Deputy Director Analysis, Canberra (Shared Doc Data Sheet & Distribution List sheet)

Army
Director General Future Land Warfare (Doc Data Sheet only)
Director Network Centric Warfare (NCW) – Army
SO1 CISEW (Force Development Group), Land Warfare Development Centre, Puckapunyal
ABCA National Standardisation Officer, Land Warfare Development Centre, Puckapunyal (emailed document data sheet)
SO (Science), Deployable Joint Force Headquarters (DJFHQ) (L), Enoggera QLD (Doc Data Sheet only)
SO (Science) - Land Headquarters (LHQ), Victoria Barracks NSW (Doc Data sheet & Exec Summ)

Air Force
SO (Science) - Headquarters Air Combat Group, RAAF Base, Williamtown, NSW 2314 (Doc Data Sheet & Exec Summ)

Intelligence Program
DGSTA Defence Intelligence Organisation
Manager, Information Centre, Defence Intelligence Organisation (PDF)
Assistant Secretary Corporate, Defence Imagery and Geospatial Organisation (Doc Data Sheet only)

Defence School of Signals
Commandant, Defence Force School of Signals, Simpson Barracks, Macleod, Vic., 3085, (Doc Data sheet & Exec Summ)

Senior Instructor Officer Command, Land-CIS training Wing, Defence Force School of Signals, Simpson Barracks, Macleod, Vic., 3085, (Doc Data sheet & Exec Summ)

Defence Materiel Organisation

Deputy CEO (Doc Data Sheet only)
Head Aerospace Systems Division (Doc Data Sheet only)
Head Maritime Systems Division (Doc Data Sheet only)
Chief Joint Logistics Command (Doc Data Sheet only)
Project Manager Air Warfare Destroyer (Doc Data Sheet only)
Director General Communications (Doc Data Sheet only)
PD JP 2072
PD SEA 1442
PD JP 2043
PD JP 2008
PD JP 2047
Land Engineering Agency – 2 copies: Library and Mr G. Lampard

Defence Libraries

Library Manager, DLS-Canberra (Doc Data Sheet Only)

UNIVERSITIES AND COLLEGES

Australian Defence Force Academy
Library
Head of Aerospace and Mechanical Engineering
Electrical Engineering (Dr M. Frater)
Hargrave Library, Monash University (Doc Data Sheet only)
Librarian, Flinders University

OTHER ORGANISATIONS

National Library of Australia
NASA (Canberra)
State Library of South Australia

OUTSIDE AUSTRALIA

INTERNATIONAL DEFENCE INFORMATION CENTRES

US Defense Technical Information Center, PDF
UK Defence Research Information Centre, PDF
Canada Defence Scientific Information Service, PDF
NZ Defence Information Centre, PDF

ABSTRACTING AND INFORMATION ORGANISATIONS

Library, Chemical Abstracts Reference Service
Engineering Societies Library, US
Materials Information, Cambridge Scientific Abstracts, US
Documents Librarian, The Center for Research Libraries, US

SPARES (5 copies)

Total number of copies:

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

Modelling of Medium Access Control (MAC) Protocols for Mobile Ad-hoc Networks

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR(S)

Raymee Chau

5. CORPORATE AUTHOR

Information Sciences Laboratory
PO Box 1500
Edinburgh South Australia 5111 Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-TR-0637	AR-013-420	Technical Note	June 2005

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
	JTW 02/098	DGICD	37	4

13. URL on the World Wide Web

http://www.dsto.defence.gov.au/corporate/reports/DSTO-TN-0637.pdf

14. RELEASE AUTHORITY

Chief, Information Networks Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS Yes

18. DEFTEST DESCRIPTORS

See your Client Liaison Librarian for DEFTEST terms

19. ABSTRACT

This technical note explains the design and modelling of two MAC protocols, Adaptive Generalized Transmission (AGENT) protocol and Collision-Avoidance Time Allocation (CATA) protocol, for mobile ad-hoc networks. These MAC models will be used to assist the analysis of performance of MAC protocols in the future Tactical Data Distribution Sub-system (TDDS), in support of Joint Project 2072 (Battlespace Communications System Land).